Universidad Carlos III de Madrid

Escuela Politécnica Superior

Sistema de Edición de Composiciones Musicales

Proyecto Fin de Carrera
Ingeniería Técnica de Telecomunicación
Sonido e Imagen

Autora: Clara Ávila Cantos
Tutor: Julio Villena Román

Septiembre de 2009

Título: Sistema de Edición de Composiciones Musicales

Autora: Clara Ávila Cantos

Tutor: Julio Villena Román

EL TRIBUNAL

Presidente:
Jaime José García Reinoso

Secretario:
Mario Muñoz Organero

Vocal:
Oscar Quevedo Teruel

Realizado el acto de defensa del Proyecto Fin de Carrera el día XX de Septiembre de 2009 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Secretario Fdo: Vocal

Resumen

El Proyecto Fin de Carrera “Sistemas de Edición de Composiciones Musicales” versa sobre el diseño e implementación de una aplicación capaz de editar partituras musicales. Todo ello se realizará desde una interfaz gráfica muy similar a las partituras originales donde se podrán seleccionar las distintas notas, figuras o silencios.

Asimismo el sistema será capaz de generar archivos MIDI con distintos instrumentos y distintos tempos. Dichos archivos se podrán reproducir gracias a la misma interfaz.

Por último, la aplicación podrá guardar en formato MusicXML todas las partituras creadas para poder ser compartidas.

El objetivo final del proyecto es diseñar un sistema completo de edición de composiciones musicales basado en estándares que permita integrar de forma sencilla módulos externos de composición y generación automática de dictados e incluso armonizador de melodías.

Agradecimientos

Enfrentarse a un proyecto es una de las cosas más difíciles que he hecho en mi vida, pero la recompensa, a cambio, ha sido una de las más satisfactorias. En este apartado quiero dar las gracias a todas las personas que han colaborado en este proyecto y, sobre todo, en esta carrera que he cursado.

Gracias a Julio Villena, por ser la persona más paciente y por haberme dado tantas oportunidades en la vida. Gracias a todos los profesores de esta Universidad que me han hecho sacar algo en claro de esta carrera y sobre todo, a los que lo han conseguido que aprenda algo de la vida.

Quiero dar las gracias a Alberto e Inmaculada, por su apoyo incondicional. A Débora y a Ana por todo ese tiempo a mi lado. A Cristina, Marta y Estefanía, por estar siempre dispuestas a escuchar. A Lucía, Celia y Sara, por convertirse en la más grata compañía. A los “Consejos de Sabios”, Juan y J. R. Camarma, por todos esos momentos.

Gracias a la familia Sonimágica: J. Crespo, Antonio, Rubén, Carlos P., Jorge H., Jesús, Mónica, Cristina L., Pablo, Carlos, Javier M., Sonia, Alex, Daniel, Álvaro, Ana B., Marian, Cristina P., Elma, Félix, Juan Luis, Ana, Juan Carlos, Aitor y toda la gente que se ha quedado por el camino. A Yoli y Antonio, eternas fuentes de sabiduría. A Nuria, amiga por y para siempre. A mí pasado, piruleta, el barrio, el córdoba, los campamentos, el instituto, Sligo y a mi presente clown, por lo que esta gente y estas experiencias me han aportado.

Gracias a todos los que me han mostrado algo bueno y también a los que me han mostrado algo malo ya que de ello también he aprendido.

Gracias a todos los que *están* y a los que *han estado*, porque *siempre estarán*.

J'ai besoin de la lune
pour lui parler la nuit.
Tant besoin du soleil
pour me chauffer la vie.
J'ai besoin de mon père
pour savoir d'ou je viens,
j'ai besoin de ma mère
pour montrer le chemin.

Manu Chao

ÍNDICE DE CONTENIDOS

TÍTULO **PÁGINA**

ÍNDICE DE IMÁGENES

FIGURA PÁGINA

1.- Introducción

"Después del silencio, lo que más se acerca a expresar lo inexpresable es la música"

Adouls Huxley

1.1.- Motivación del Proyecto

¿Por qué plantear el desarrollo de un Sistema de Edición de Composiciones Musicales? ¿Cuál es su hueco y su utilidad en una esfera, la musical, que precisamente se ha caracterizado por una relativa independencia respecto a las nuevas tecnologías, al menos, en lo que a su aprendizaje se refiere? La respuesta a estos interrogantes es la que dará sentido al desarrollo de este Proyecto Fin de Carrera que aquí se describe.

Los principales teóricos de nuestra época, coinciden en definir a nuestra sociedad como la Sociedad del Conocimiento y las Nuevas Tecnologías. Y es que no cabe ninguna duda de que el desarrollo e implantación de dichas tecnologías ha supuesto un antes y un después en nuestra forma de vivir y entender la realidad.

No resulta de extrañar, por tanto, que en los últimos años hayan sido múltiples los esfuerzos por parte de las instituciones formativas para hacer realidad la conjunción entre enseñanza y nuevas tecnologías. Así, con un ritmo lento pero seguro, poco a poco estas tecnologías han ido abriéndose paso dentro de las aulas, como una herramienta cada vez más indispensable en la transmisión y adquisición del conocimiento.

Sin embargo, aún queda mucho camino por recorrer y, en este sentido, resultan significativas las importantes diferencias de ritmo ente los distintos campos del conocimiento. Quizá no es arriesgado afirmar que una de las disciplinas donde más retraso se está dando en la incorporación de las nuevas tecnologías al aprendizaje sea la música.

Precisamente es este el motivo que guía el proyecto: ser capaces de ofrecer una herramienta didáctica que acerque al estudio de la música las nuevas tecnologías informáticas.

1.2.- Objetivos

Todo estudiante de música sufre ciertas dificultades en su proceso de aprendizaje. Aprender la grafía, entender los tiempos musicales, diferenciar los instrumentos, las distintas notas, etc. Toda esta base musical debe ser incorporada, y la tarea no es sencilla.

El objetivo del proyecto es el diseño e implementación de una herramienta básica de edición de composiciones musicales orientada a estudiantes de música. Dicha aplicación deberá poder realizar ediciones de composiciones musicales basada en estándares que permita integrar de forma sencilla módulos externos de composición y generación automática de dictados e incluso armonizador de melodías.

En este sentido, esta aplicación se plantea como una herramienta de apoyo, que permita un aprendizaje autónomo por parte del estudiante de los principales contenidos musicales. Pero a la par, la aplicación tiene otra virtud igual de importante: si un estudiante de música se familiariza desde los primeros momentos del aprendizaje con una herramienta de edición, en un futuro tendrá menos problemas a la hora de acercarse a un ordenador en busca de ayuda.

Poco a poco, el concepto de frialdad que una buena parte de los músicos asocia a las aplicaciones informáticas, desaparecería gracias a una familiarización progresiva con las mismas.

Puesto que el público al que va dirigido son estudiantes de música, uno de los principales requisitos que debe cumplir el diseño de esta herramienta didáctica será la facilidad en el manejo de la misma, como elemento clave en la consecución de su éxito.

Otro objetivo de diseño que busca que la aplicación, es la modularidad. De esta forma se le podrán incorporar bloques externos o “plugins” en el futuro de forma fácil. Esto enriquecerá el contenido y las funcionalidades del sistema creando, por tanto, una completa y potente herramienta didáctica para estudiantes de música.

Desde un principio se planteó que el sistema fuera multiplataforma y, por tanto, se pudiera utilizar bajo distintos sistemas operativos como Windows, Linux, Mac, etc. Para resolver este requisito se ha utilizado como lenguaje de programación Java, ya que es el que mejor se adapta a todas las plataformas anteriormente mencionadas.

1.3.- Estructura de la Memoria

Se incluye a continuación una breve explicación de la estructura expositiva que se va a seguir a lo largo de la presente memoria, con el objetivo de situar al lector en las distintas partes que la componen.

Para clarificar los principales conceptos e ideas que constituyen la base teórica de la Memoria del Sistema de Edición de Composiciones Musicales se comenzará en el capítulo 2 de “Estado del Arte”.

En este capítulo se hará un breve repaso de las principales tecnologías existentes en la actualidad, inscribiéndolas en su contexto histórico de surgimiento y desarrollo hasta llegar al momento presente.

A continuación, en el capítulo 3, se expondrá el diseño del sistema, que constituye la auténtica guía de trabajo desde la que se empezará a construir la aplicación. En este apartado quedarán explicitados los objetivos en los que se basa el proyecto, con el fin de asegurar su cumplimiento en el desarrollo posterior.

En este capítulo será necesario hacer alusión también a la relación entre todos los componentes del sistema, su interactuación y la relevancia que tienen.

Posteriormente llegará el turno de exponer con detalle, en el capítulo 4, todo el proceso de implementación, donde se explicará paso a paso la creación del sistema. Será necesario especificar en él la relación entre las variables, los métodos, las clases y cómo, gracias a la programación, se conseguirán los objetivos.

Por último, en el capítulo 5, se expondrán las conclusiones y las perspectivas que se abren para posibles trabajos futuros sobre la base de la aplicación desarrollada. Se trata, por tanto de una pequeña retrospectiva de la ardua tarea que supone la creación de un proyecto, que se completará con una mirada hacia delante.

Se encontrará, a continuación, en la sección de Anexos, un manual de usuario con una visión global de la aplicación creada y de cuál es su funcionamiento.

2.- Estado del Arte

2.1.- Nociones Básicas de Música

"Cuando no me ve nadie, como ahora, gusto de imaginar a veces si no será la música la única respuesta posible para algunas preguntas."

Antonio Bueno Vallejo

La música, desde la existencia del ser humano, ha sido una forma de expresarse. Se define "música" (del latín musĭca, y ésta del griego μουσική) como "un sonido grato al oído" [8].

Por tanto, medir y poder expresar gráficamente la música es hablar de la gramática de otro tipo de expresiones. Antes de que se comience razonando por qué tiene tanta importancia la música en relación con este proyecto, se intentará hacer más comprensible algunos conceptos básicos que han sido utilizados.

Se define pentagrama como "renglonadura formada con cinco rectas paralelas y equidistantes, sobre la cual se escribe la música" [8]. Al principio de cada pentagrama encontrará la clave que servirá para determinar el nombre de las notas. Para este programa se utiliza la clave de Sol, ésta dice que "la nota Sol se encuentra en la segunda línea del pentagrama" [15].

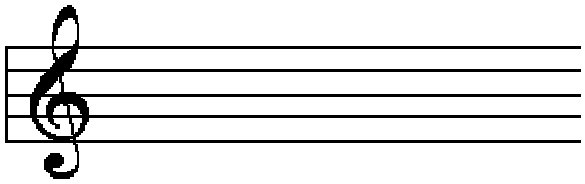

Figura 1: Pentagrama con Clave de Sol

Los sonidos musicales están representados por las notas. La altura sonora se representa situando estos signos en las diferentes líneas y espacios del pentagrama. Existen siete notas musicales (Do, Re, Mi, Fa, Sol, La, Si), que ordenados de grave a agudo forman la escala musical.

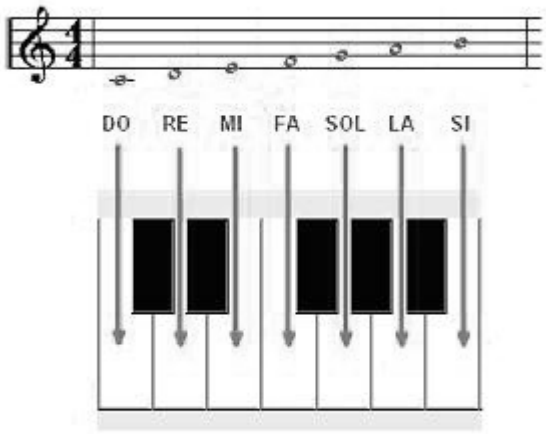

Figura 2: Escala de notas musicales y su representación en un piano

La distancia entre cada nota es de un tono, a excepción de la distancia entre Mi y Fa, y la distancia entre Sí y Do.

Existe una notación alternativa en el mundo anglosajón, según se muestra en la figura siguiente:

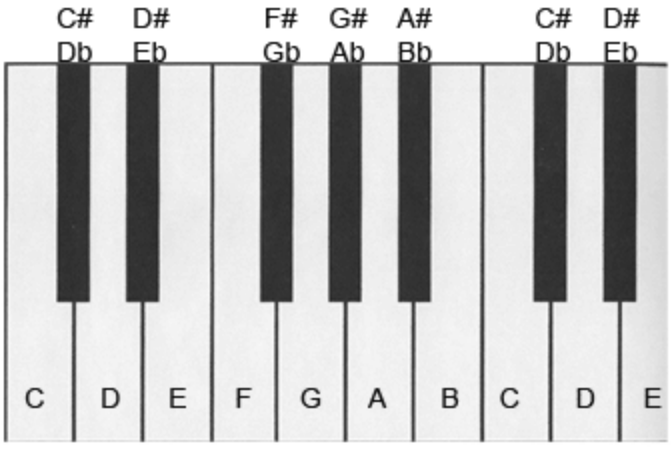

Figura 3: Notación alternativa para la escala musical

En ausencia de nota musical se encuentran los silencios. Éstos tienen la misma duración que las figuras musicales sin ninguna tonalidad.

Las figuras son signos que identifican una duración. Estas son principalmente: redonda, blanca, negra y corchea. Cada una tiene duración doble a su antecesora y mitad que su predecesora.

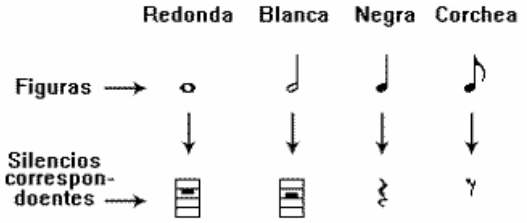

Figura 4: Figuras y silencios correspondientes

Cuando se habla del ritmo o cadencia de una pieza musical, se debe definir el compás como el "signo que determina el ritmo en cada composición o parte de ella y las relaciones de valor entre los sonidos".

Los compases se indican por medio de dos cifras, que se representan en forma de fracción.

En los compases de subdivisión binaria, el numerador representa el número de tiempos que tendrá el compás. Los compases más comunes tienen 2, 3 o 4 tiempos. El denominador representa la figura que llenará un tiempo del compás. Por convención 4= negra; 8= corchea.

En la siguiente figura se muestran los tipos de compases en este caso binario, ternario y cuaternario:

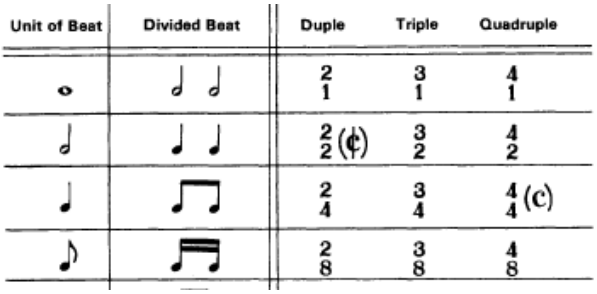

Figura 5: Compases Simples

Por ejemplo, en un compás de 2/4, cada compás tendrá dos pulsos, y el denominador 4 indica que la unidad será la negra. Esto significa que cada compás tendrá dos negras.

Para separar un compás de otro se indicará con una línea vertical. Esto sucederá cuando se haya completado un compás y ayudará a saber en qué momentos hay que poner acentos musicales (generalmente al principio de cada compás).

Para finalizar con esta relación de conceptos musicales se definirá sostenido y bemol. “Sostenido es el signo (#) que representa la alteración del sonido natural de la nota o notas a que se refiere en medio tono superior” [29]. “Bemol es el signo (ь) que representa esta alteración del sonido natural de la nota o notas a que se refiere” [29].

De esta forma, al dibujar una nota con sostenido o con bemol se le está aplicando una modificación tonal de medio tono superior o inferior.

En la siguiente imagen se ve un claro ejemplo de ello:

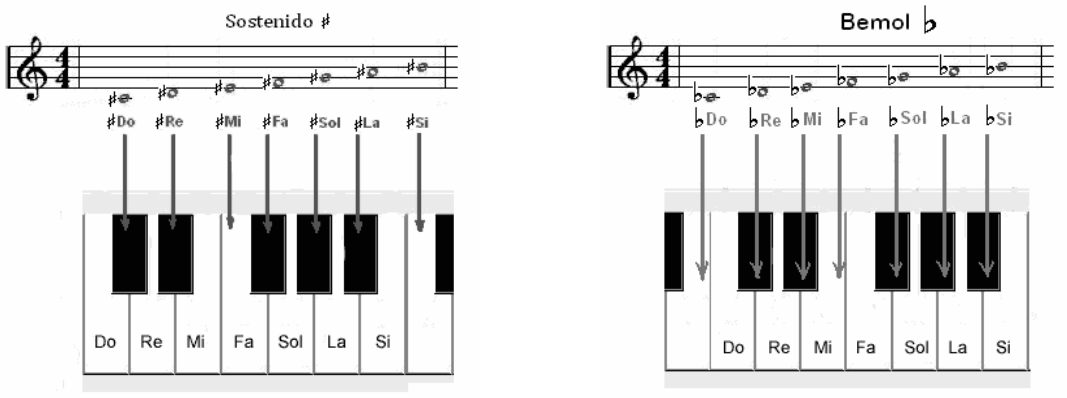

Figura 6: Sostenidos y Bemoles

2.2.- Relación entre la Informática y la Música

"En el pensamiento científico siempre están presentes elementos de poesía. La ciencia y la música actual exigen de un proceso de pensamiento homogéneo."

Albert Einstein

Gran parte de la relación entre la informática y la música tiene mucho que ver entre la teoría musical y las matemáticas.

El primer computador capaz de reproducir música fue CSIRAC "diseño de Trevor Pearcey y Maston Beard, Australia 1949" [20].

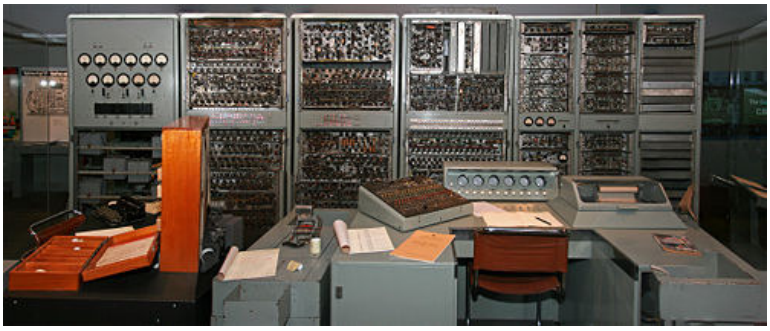

Figura 7: CSIRAC, expuesta en el Museo de Melbourne

Entre 1950 y 1951, la CSIRAC fue usada para reproducir música, el primer uso conocido de un computador digital para este propósito. Esa música nunca fue grabada, pero ha sido reconstruida con exactitud. No obstante esta máquina solo era capaz de reproducir un repertorio estándar, no fue usada para extender el pensamiento musical o generar composiciones.

La máquina no tenía sistema operativo. "En 1960, Geoff Hill desarrolló un lenguaje de programación similar a las formas primerizas de BASIC" [20].

Cuando se habla de generar música, hay que remontarse a Febrero de 1951 (Universidad de Manchester) donde se creó la primera máquina generadora: “Ferranti Mark 1” [5] (un mes después se creó UNIVAC 1, similar a la anterior).

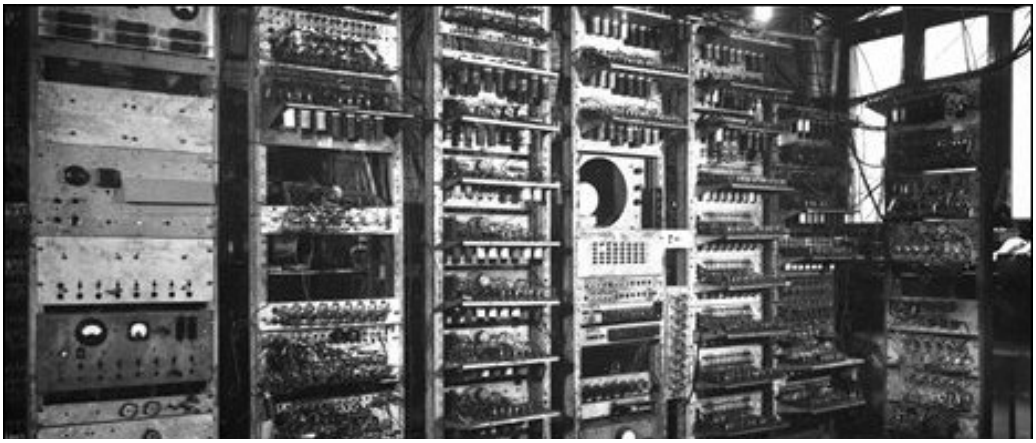

Figura 8: Ferranti Mark 1, lado izquierdo

Ferranti Mark 1 tenía muchas mejoras frente a CSIRAC como la posibilidad de tener un control de retorno a los operarios o la opción de alterar la frecuencia de las melodías. “También se podían grabar las melodías gracias a la tecnología de la BBC y la programación de manos de Christopher Strachey”. [5]

Dejando de lado estás dos máquinas se pasará a hablar de los sintetizadores musicales como es el caso del Vocoder. El funcionamiento es muy simple, una fuente de sonidos musicales se usa como portadora, en lugar de extraer la frecuencia fundamental. Por ejemplo, se puede usar el sonido de una guitarra como la entrada del banco de filtros.

Los pioneros de la música electrónica, “Wendy Carlos y Robert Moog” [1] desarrollaron uno de los primeros vocoders verdaderamente musicales. Un dispositivo de diez bandas.

Figura 9: Vocoder actual, marca KORG

Ejemplos famosos de su uso son la banda sonora de la película “La Naranja Mecánica” (Stanley Kubrik) donde el vocoder “canta” la parte vocal de la Novena Sinfonía de Ludwig van Beethoven.

Los primeros programas musicales generalmente no se podían ejecutar en tiempo real, podrían tardar en finalizar horas o días, la gran mejora de los chips digitales y los microprocesadores abrieron la puerta a la generación musical en tiempo real.

A principios de los años noventa se pudo alcanzar el punto de la generación musical en tiempo real utilizando programas y algoritmos mucho más sofisticados.

Estos avances en la potencia de los ordenadores han mejorado en gran medida la forma en la que la música es generada y mostrada en ordenadores. Los actuales microprocesadores son suficientemente potentes para crear sistemas de audio muy sofisticados gracias a una amplia variedad de algoritmos de programación.

Todo esto no sería posible sin una cantidad de inventos que soportan la creación musical: sintetizadores, mezcladores digitales, unidades de efectos. Todos ellos se han convertido en un apoyo, más para el uso digital que para el analógico, convirtiendo esta actividad en la norma y no en la excepción.

2.2.1.- La Música y la Inteligencia Artificial

Se denomina Inteligencia Artificial (IA) a la “rama de la ciencia informática dedicada al desarrollo de agentes racionales no vivos” [8]. Es decir, la IA es la “disciplina que se encarga de construir procesos que al ser ejecutados sobre una arquitectura física producen acciones o resultados que maximizan una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura” [13].

La investigación musical en el aspecto de implementación de la teoría explicada en el apartado 2.1 y práctica musicales necesita los avances en IA. Se exige una mayor comprensión de los procesos mentales implicados en las tareas musicales; por otra parte, a los sistemas de procesamiento musical se les exige características en niveles de complejidad superiores a las ofrecidas por los sistemas tradicionales de música a través de ordenador.

Es decir, los ordenadores se podrían comportar como "máquinas inteligentes". De esta forma podrían ser capaces de representar el conocimiento musical de tal manera que puedan ser manipulados para el cumplimiento de determinadas tareas en los campos de la composición, audición, análisis, ejecución, adiestramiento y procesamiento digital del sonido. Cada una de estas áreas implican determinados problemas complejos a resolver mediante distintas aproximaciones en IA.

La IA puede beneficiar mucho a la investigación musical puesto que la música es un paradigma experimental complejo.

Las contribuciones mutuas de la IA y la música son fuentes de interesantes debates entre los investigadores en IA musical: algunos de ellos intentan demostrar la relación simbiótica entre la música y la IA, mientras que otros expresan su asombro ante estas tesis. Este debate abierto constituye una prometedora plataforma para un futuro desarrollo de esta joven disciplina.

Un programa inteligente de ayuda al compositor debería permitir una representación estructurada de las entidades musicales en diferentes niveles de abstracción. En ciertas situaciones un compositor necesita trabajar con todos los detalles de bajo nivel de un objeto musical; en otras ocasiones, por ejemplo cuando se planea la estructura global de una parte de la obra, el mismo objeto musical debería ser representado en otro nivel de abstracción, como una entidad simbólica de alto nivel, contemplando sólo algunas de sus propiedades y sus posibles relaciones con otros objetos musicales. [28].

Este caso es el que más se aproxima al Sistema de Edición de Composiciones musicales ya que no se busca que el compositor conozca los detalles de bajo nivel sino que la aplicación sea útil para la finalidad planteada.

2.3.- Tecnologías Utilizadas

"El pensamiento, cuanto más puro, tiene su número, su medida, su música".

María Zambrano

2.3.1.- Soportes Musicales

Desde que el sonido pudo ser registrado y posteriormente reproducido, la relación entre las personas y la música cambió drásticamente gracias al nuevo enlace intermedio: el aparato reproductor electrónico.

De esa forma, se pasó por distintos soportes (discos, casetes) hasta llegar a los soportes electrónicos que revolucionaron los procesos de reproducción y distribución musical, e involucraron en el proceso a ingenieros, técnicos, diseñadores y un largo etcétera.

El primer soporte que fue comercializado fue el acetato o disco de vinilo. Ha sido el medio de distribución con más años en el mercado.

Tiene una buena calidad de sonido pero tras muchos usos se desgastaban los discos (ya que para su reproducción se necesita una aguja).

El siguiente de los soportes inventados fue las grabadoras y reproductores de casetes. Usaban cintas magnetofónicas que, mediante la manipulación magnética de partículas distribuidas en su superficie, plasmaba señales de audio. No obstante, el problema de perder su contenido con el tiempo no se había solucionado.

La industria evolucionó y consiguió solucionar los problemas del paso del tiempo con los Compact Disc (Philips, 1982) y los DVD's [30]. Estos formatos alternativos funcionan a partir de un haz de luz que lee una información digital (codificada con números binarios), eliminando así el contacto directo que desgasta las superficies y permitiendo reducir el tamaño del soporte.

Dentro de esas grandes mejoras todavía quedaba mucho en lo que trabajar: nuevas formas de distribución y de compresión de datos. De aquí surgen los Reproductores de Audio Digital, que son dispositivos capaces de almacenar, organizar y reproducir archivos de audio digital. Suelen llamarse reproductores MP3 por ser el formato de audio digital más utilizado en estos dispositivos (pero no el único ya que suelen reproducir Ogg, AAC, WMA, etc.)

Este tipo de reproductores pueden basarse en almacenamientos en memorias Flash (almacenan los ficheros en memorias internas o externas, como tarjetas de memoria) o basados en disco duro (gran capacidad, de 1,5GB a 100Gb, por ejemplo iPod de Apple, Zen de Creative, etc.)

Figura 10: Soportes de Audio Descritos: (1) Disco Vinilo, (2) Casete, (3) Compact Disc, (4) Reproductor de Audio Digital

2.3.2.- Estándares de Archivos de Audio Digital

Los archivos de audio digital son el resultado de digitalizar una señal eléctrica que representa una onda sonora. El proceso de digitalizar consta de varias partes. En la primera, la señal analógica es muestreada (tomando la amplitud de la señal eléctrica a intervalos regulares de tiempo). Posteriormente, es cuantificada (convirtiendo el valor de las muestras en un valor entero de rango finito y predeterminado). Para finalizar, la señal es codificada (los valores anteriores son traducidos al sistema binario).

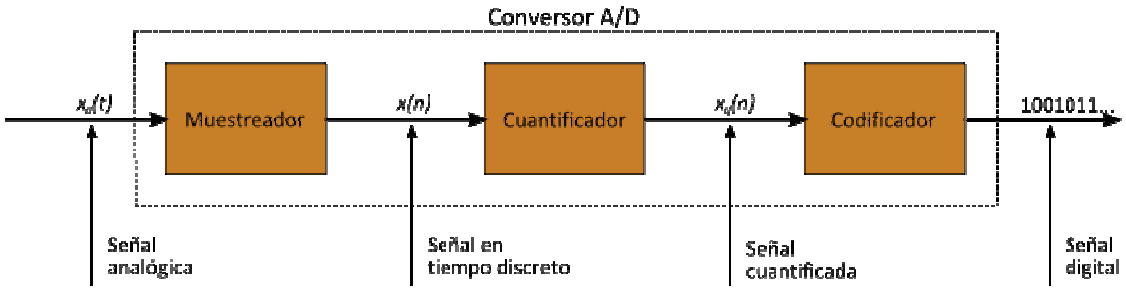

Figura 11: Diagrama de bloques de la cadena de Digitalización

Es en este proceso donde se lleva a cabo la compresión de la señal que consiste en la reducción del tamaño de los archivos de audio y puede hacerse basándose en algoritmos de compresión sin pérdida o algoritmos de compresión con pérdida.

Sobre este tema se podría hacer un gran desarrollo ya que el proceso de digitalización y compresión del audio es motivo de muchos estudios, no obstante, se expondrán a continuación los formatos de audio digital más importantes e interesantes para este proyecto: el MPEG-1 Audio Layer 3 y MIDI.

MPEG-1 Layer 3, más conocido como MP3, es un formato de audio digital comprimido con pérdida desarrollado por el Moving Picture Experts Group (MPEG, grupo especializado en crear y mejorar los estándares para compresión de audio y video digitales). El mp3 estándar es de 44 KHz, un bitrate de 128 kbps y un ratio de compresión de 12 a 1, o sea menos de una décima parte de lo que ocupa una pista de CD [32].

La compresión se basa en la reducción del margen dinámico irrelevante, es decir, el oído humano sólo percibe un rango acústico de frecuencias que van desde los 20Hz hasta los 20KHz (siendo entre los 2KHz y 4KHz el rango de mayor percepción al oído), mientras que los archivos de sonido sin compresión registran información por fuera de tales límites. A este respecto, lo que hace la tecnología MP3 es desechar todos aquellos sonidos que se encuentran por debajo de los 20Hz o por encima de los 20KHz.

A principios de 2002 otros formatos de audio comprimido como Windows Media Audio y Ogg Vorbis empiezan a ser incluidos en programas, sistemas operativos y reproductores. Esto pareció el declive del formato MP3 ya que los otros formatos mencionados son de mayor calidad y que el formato MP3 tiene patente, es decir, la comunidad no puede seguir mejorándolo y puede obligar a pagar por utilizar algún códec. No obstante, a día de hoy sigue siendo el más usado [32].

2.3.3.- MIDI

MIDI, siglas de Musical Instrument Digital Interface, es el lenguaje que utilizan actualmente muchos instrumentos (ordenadores, sintetizadores, controladores y otros dispositivos musicales electrónicos) para comunicarse entre ellos, enviar y recibir datos y sincronizarse. Se trata de un protocolo industrial estándar.

Esta nueva tecnología nació en Los Ángeles, en 1981 en un documento dirigido a la Audio Engineering Society por Dave Smith (presidente de Sequential Circuits). No obstante, hasta 1983, no se publicó la primera especificación [6].

Esta información define diversos tipos de datos como números que pueden corresponder a notas particulares, números de patches de sintetizadores o valores de controladores. Gracias a esta simplicidad, los datos pueden ser interpretados de diversas maneras y utilizados con fines diferentes a la música.

Se debe aclarar que MIDI no transmite señales de audio, sino que es una especie de "partitura" que contiene las instrucciones en valores numéricos (0-127) sobre cuándo generar cada nota de sonido y las características que debe tener; el aparato al que se envíe dicha partitura la transformará en música completamente audible [21].

En la actualidad la gran mayoría de los creadores musicales utilizan el lenguaje MIDI a fin de llevar a cabo la edición de partituras y la instrumentación.

Se debe distinguir entre dos modos: monofónico (sólo puede reproducir una nota) y polifónico (puede reproducir varias notas simultáneamente, como los acordes de un piano).

La estructura de los datos se distribuye de manera distinta ya que el byte MIDI está compuesto por diez bits según el estándar. El primero es el bit de inicio (start bit, que siempre es 0) y el último el bit de terminación (stop bit que siempre es 1).

Esto con el fin de que los dispositivos MIDI puedan llevar la cuenta de cuantos bytes se han enviado o recibido. Los ocho bits restantes contienen los mensajes MIDI.

Existen dos tipos de bytes: de estado (primer bit es un 1) y de información (primer bit es un 0). Al generar un mensaje MIDI siempre se enviará un byte de estado (por ejemplo, “activar nota”), que puede estar seguido de cierta cantidad de bytes de datos (diciendo qué nota es la que se activa).

2.3.4.- MusicXML

Actualmente, es normal que cualquier campo de la vida humana se trate de resolver las necesidades de comunicación a través de Internet. La música no es la excepción.

Una de la novedades básicas que Internet introdujo al campo musical, fue la modificación de las formas de estructuración de la información al presentarla de manera no jerárquica, sino interactiva e hipertextual; alentando el reconocimiento de que al utilizar estos nuevos canales y aplicaciones, se está dirigiendo a un público que posee, además de referencias musicales, cierto grado de información tecnológica.

La ventaja de Internet es que pone al alcance de los usuarios información sobre música que en otros tiempos tardaba meses, o quizás años, en obtenerse.

MusicXML es un formato digital para el intercambio y la distribución de datos musicales. El objetivo es la creación de un formato universal para notación musical similar al rol que desempeña el formato MP3 con la música grabada. La información musical debe ser útil por otros programas.

MusicXML está basado en dos formatos musicales teóricos: MuseData, desarrollado por Walter Hewlett en la Universidad de Stanford y el formato Humdrum, creado por David Huron en la universidad de Ohio [23].

La primera versión beta de MusicXML está basada prácticamente en una actualización XML del formato MuseData que incorpora algunos conceptos clave de Humdrum.

Figura 12: MusicXML

Desde que ambos formatos fueron creados para trabajar con música clásica y folclórica, MusicXML ha sido extendido para poder soportar música contemporánea. Asimismo sirve como formato de distribución de hojas digitales musicales y como formato de intercambio.

Existen muchos programas que facilitan en tratamiento digital de la música, desgraciadamente compartir dicha información musical entre ellos es difícil ya que no están estandarizados. Esto es un problema real ya que estos programas no hacen todas las funciones igual de bien y tener que reintroducir toda la información musical para cada programa se ha convertido en un problema importante para aquellas personas que no usan un solo programa musical.

Antes de MusicXML la única notación o protocolo de intercambio de datos que estuviera estandarizado era MIDI. MIDI es un buen protocolo pero, tiene ciertas carencias musicales ya que, por ejemplo, no diferencia entre un Fa sostenido y un Sol bemol (pese a que sonoramente son iguales no lo son a nivel de notación), tampoco diferencia entre repeticiones, compases y otros aspectos relativos a la notación musical.

MusicXML contiene dos tipos principales de elementos. Una serie de elementos son los encargados de representar cómo sonará la pieza musical. Otros se encargan de cómo debe representarse en un pentagrama, éstos son los encargados de crear el documento final de MusicXML.

El DTD de MusicXML y el XSD son libres pero no abiertos, como en casos anteriores se tiene la opción de usarlo pero no de mejorarlo. La licencia de utilización, de Recordare, está modelada y adaptada a World Wide Web Consortium (W3C). [23]

2.3.5.- Lenguajes de Programación

"Programar es elaborar programas para la resolución de problemas mediante ordenadores" [8]. "Un lenguaje de programación es un conjunto de símbolos y reglas que sirve para controlar el comportamiento físico y lógico de una máquina.

Se busca diseñar un lenguaje artificial que pueda expresar a las máquinas el comportamiento que se busca en ellas, bien sea expresando algoritmos o a modo de comunicación humana" [14].

Cuando se programa se especifica de manera precisa sobre qué datos debe operar un ordenador, se habla de su tratamiento, almacenamiento o transmisión. Todo ello es comunicado mediante un lenguaje relativamente cercano al lenguaje humano.

Posteriormente el procesador del ordenador entenderá y actuará según lo indicado por el programa escrito en lenguaje fijo (lenguaje máquina).

Para que un programa escrito en otro lenguaje pueda ser ejecutado tienen que pasar dos cosas: o bien que el programa adapte las instrucciones conforme las va encontrando (proceso de interpretar hecho por programas intérpretes), o bien que el programa se traduzca del lenguaje de programación al lenguaje máquina por medio de un compilador (proceso de compilar) [18].

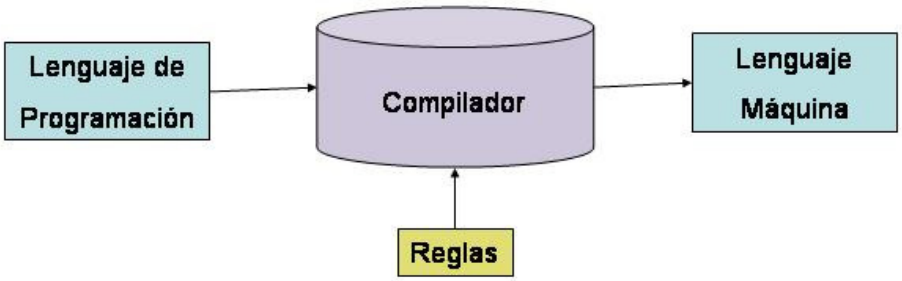

Figura 13: Esquema de funcionamiento de un compilador

Según el nivel de abstracción se pueden clasificar cuatro tipos de lenguaje de programación. El primero sería *el lenguaje máquina* que es legible directamente por la máquina (las instrucciones son cadenas binarias). El segundo serían los *lenguajes de bajo nivel* que se acercan al funcionamiento de un ordenador (serían el código máquina y el lenguaje ensamblador).

El tercero sería el lenguaje de medio nivel ya que tiene ciertas características similares a los lenguajes de bajo nivel, pero al mismo tiempo ciertas cualidades que le hacen cercano al lenguaje humano (por ejemplo el lenguaje C). Para finalizar se habla de los lenguajes de alto nivel, son los más fáciles de aprender porque se basan en elementos de lenguajes naturales como el inglés.

Estos lenguajes también pueden clasificarse por el paradigma de programación en cuatro grupos: paradigma imperativo (por órdenes como C o BASIC), paradigma funcional (familia de lenguajes LISP, en concreto Écheme), paradigma lógico (como PROLOG) y por último paradigma orientado a objetos. El uso de este último se popularizó a principios de la década de 1990. Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos [11].

La Programación Orientada a Objetos (POO u OOP) usa objetos y sus interacciones para diseñar aplicaciones y programas para el ordenador. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento.

Un ejemplo de Programación Orientada a Objetos es el lenguaje Java desarrollado por Sun Microsystems a principios de los años noventa.

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un bytecode. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución.

Es decir, Java no es un lenguaje compilado o interpretado, son las dos cosas a la vez ya que primero se compila, y el resultado, el bytecode se interpreta [17].

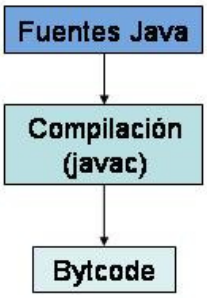

Figura 14: Desarrollo de Programas en Java

Java es un lenguaje muy moderno, a pesar de ello, ha ganado muchos adeptos rápidamente por muy diversas razones, una de ellas es la posibilidad de añadir programas a una página web.

A ello se le une las bibliotecas ya definidas que proporciona el lenguaje y que el programador puede utilizar sin tener que hacerlas de nuevo.

2.3.6.- JMusic

Cuando se busca la aplicación de todas estas técnicas para la música, surgen conceptos nuevos como los lenguajes de programación musicales. Estos son lenguajes de programación específicos del sector del sonido y la producción musical o síntesis.

Para este caso concreto, se hablará de una librería externa de Java: jMusic. Éste es un proyecto diseñado para dar a los compositores y programadores una librería con gran cantidad de herramientas de procesamiento y composición. Puede ser utilizada para generar música, sintetización de instrumentos y análisis musical.

Fue creado por Andrew Sorensen y Andrew Brown y publicada en Noviembre de 1998. En palabras de sus creadores, jMusic es una librería de programación escrita para músicos que utilizan el lenguaje Java para desarrollar sus aplicaciones [16]. Este proyecto quiere lograr una librería que sea suficiente para nuevos programadores pero sofisticada para dar a los compositores herramientas suficientes para el trabajo real.

Está diseñado para ser usado como un medio de composición, centrado más en músicos que en programadores.

No obstante, muchas personas consideran jMusic como un API muy útil dentro del software musical, y en concreto, ventajoso para la creación de instrumentos digitales.

jMusic está diseñado para ayudar en el proceso de composición dando una estructura de programación en el ámbito del sonido, también para análisis y educación musical.

Es un paquete de Java, programado en dicho lenguaje y no en un meta-lenguaje. Es decir, que las ventajas de Java (potencia, plataforma independiente) se mantienen. Cuanto más sepa usar el programador Java, mejor podrá aprovechar jMusic.

Es un API Musical libre (bajo la licencia de GNU) que sirve como herramienta de construcción y composición musical para distintos instrumentos. Este lenguaje utiliza una estructura de datos para música que está asociada a una interfaz gráfica básica.

Es un lenguaje fácil de aprender y de gran potencia. Su comprensión es fácil ya que respeta las convenciones de la música tradicional. Mostrar los resultados de jMusic con otra aplicación es muy sencillo ya que utiliza como medio para importar y exportar ficheros con el protocolo de comunicación MIDI.

2.4.- Editores de Partituras Existentes

"La música es la aritmética de los sonidos, como la óptica es la geometría de la luz."

Claude Debussy

Es importante analizar las herramientas existentes en el mercado para, a partir de ellas, ver qué se puede mejorar. De esta forma se resumirán las aplicaciones similares existentes en el mercado.

2.4.1.- Vivaldi Studio

Vivaldi [31] es la firma desde la cual se comercializa el software de Allegroassai, responsable entre otros programas, de Amadeus y Opus. Vivaldi Plus sigue la misma línea de editores, pero con las mejoras que implica el tiempo de desarrollo. Sus principales funciones son:

- Posibilidad de definir los márgenes de cada página.
- Posibilidad de insertar hasta 16 pentagramas por sistema.
- Posibilidad de inserir pentagramas simples, dobles, para órgano, para voz y para piano.
- Ocho voces por pentagrama.
- Posibilidad de centralizar pentagramas y sistemas en modo automático o manual.
- Inserción, borrado o cambio de símbolos directamente con el mouse.
- Ventana "Graphic Mixer" para regular volumen, cambio de programa, canal, voz, instrumento, porta, accionar el playback y la grabación.
- Puerta y canal MIDI.
- Salida MIDI a cualquier interface o periférico.
- Posibilidad de salvar tablas de sonidos programados.
- Posibilidad de imprimir partituras y escanearlas.
- Inserción de texto con posibilidad de seleccionar fuente u tamaño.

- Seis tipos de barra de compás (simple, doble, repetición, final, puntada o invisible) que pueden ser posicionadas en un solo compás o en todo el sistema.
- Posibilidad de inserir armaduras de clave en cualquier pentagrama y de impostar la transposición automática.
- Posibilidad de arrastrar las alteraciones con el mouse.
- Crescendos y diminuendos que continúan automáticamente en el pentagrama o sistema sucesivo.
- Símbolo de Pedal que continúan automáticamente en el pentagrama o sistema sucesivo.
- Ligaduras con cuatro puntos de control que continúan automáticamente en el pentagrama o sistema sucesivo.
- Ajustamiento manual o automático de los corchetes de grupos irregulares.
- Espaciado automático o manual de notas.
- Función autobarras.
- Posibilidad de modelar y diseñar ligaduras.
- Transposición diatónica y cromática.
- Posibilidad de cambiar el valor de una nota ya presente en la partitura.
- Ejercitar una parte silenciándola en la partitura, ralentando el tempo o repitiendo un pasaje.

A continuación se muestra una imagen relativa a la aplicación:

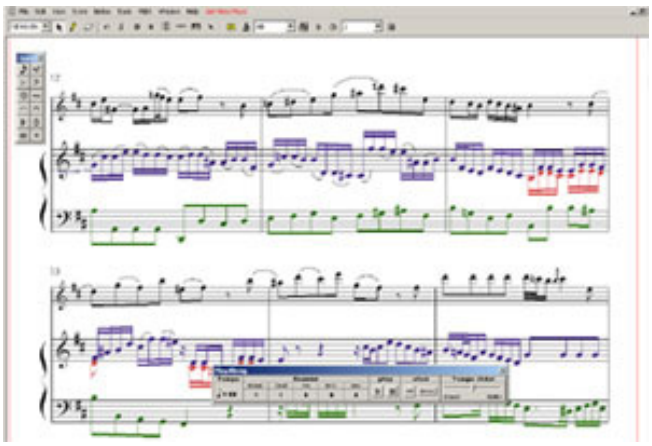

Figura 15: Ventana de Edición Principal de Vivaldi Plus

2.4.2.- Finale

Finale [10] es una aplicación pensada especialmente para compositores, profesores y estudiantes de música.

Cuenta con una completa interfaz, un reproductor integrado donde escuchar tus composiciones, soporte para dispositivos MIDI, posibilidad de grabación en WAV, MP3 o AIFF.

Finale permite introducir notas mediante el uso de instrumentos musicales basados en MIDI, como por ejemplo teclados electrónicos, o bien directamente con el ratón. Puede crear acompañamientos con soporte para partes enlazadas, repeticiones, etc. También incluye un "Asistente de ejercicio" que puede servir a los educadores musicales como apoyo para enseñar composición. Finale incluye más de sonidos de instrumentos, también tiene capacidad para grabar o importar pistas vocales e instrumentales, etc.

Cuenta con un módulo, SmartMusic, el generador de acompañamiento musical de Finale, con más de 100 sonidos de instrumentos de Tapspace Virtual Drumline.

Otras de sus características son:

- Rápido e intuitivo en la entrada, creación, copia, y movimiento de dinámicos, indicaciones tempo, y otros textos y marcaciones.
- Más de 300 sonidos Garritan.
- Más de 100 sonidos del Tapspace Virtual Drumline.
- Reproductor de Aria de Garritan.
- Acompañante SmartMusic.
- Soporte ASIO: Reduce latencia para usuarios de Windows.
- Edición Multipágina: Edición mas rápida y manejo de puntuaciones.

- Mejoras en el Human Playback: Más Simple, Más Fácil, Más accesible.
- Soporte completo para instrumentos VST/AU para posibilidades ilimitadas de reproducción.

La imagen correspondiente a la aplicación anteriormente descrita será:

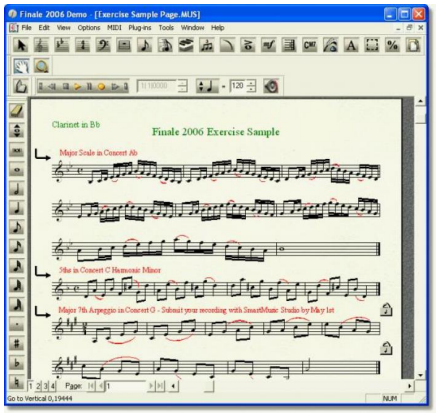

Figura 16: Ventana de Edición Principal de Finale

2.4.3.- Encore

Encore [14] es una aplicación con buena capacidad de trascripción y edición. Rápida, precisa y fácil de usar, Encore le permite crear impresiones usando hasta 64 pentagramas por separado.

Si usted utiliza directamente en Encore o transcribe sus archivos MIDI, podrá obtener la notación exacta. Puede extraer partes, para incorporar los diferentes instrumentos y reproducir dichas creaciones. Encore muestra gráficamente y reproduce marcas de dinámica, múltiples terminaciones, marcas de pedal, o cualquier controlador MIDI.

Características:

- Más de 30 plantillas de usuario y sin límite de plantilla creada.
- Auto espaciado cuando arrastra las notas, barras y sistemas.
- Opción simplificada para las alteraciones.

- Importación y exportación de archivos MusicXML.
- Insertar los datos de dinámica de MIDI, datos de controlador, y tempos.
- Ligado a un grupo de notas automáticamente.
- Colocación de los objetos en cualquier lugar de la página haciendo clic en el ratón.
- Pantalla de reproducción y repetición de estructuras anidadas, incluyendo variables de barras y líneas múltiples terminaciones.
- Explicación de los signos son: notas, silencios, alteraciones, etc.
- Introduzca el texto o la letra en cualquier lugar de la página en cualquier tipo de letra, tamaño o estilo.
- Añadir a la guitarra diagramas con los acordes del traste.
- Reproduce los archivos MIDI teniendo en cuenta la dinámica, las articulaciones, las marcas de pedal y las repeticiones.
- Versión para Mac y Windows con el mismo formato, los archivos sirven para ambas plataformas.
- Soporta el estándar MusicXML y archivos MIDI.
- El usuario puede definir valores predeterminados en el espaciamiento de los pentagramas, claves, letras de canciones, acordes, clave, y metro, colocar los encabezados y pies de página en cada página.
- Imprime una partitura, páginas individuales o partes de una partitura.

La ventana principal de la aplicación será la siguiente:

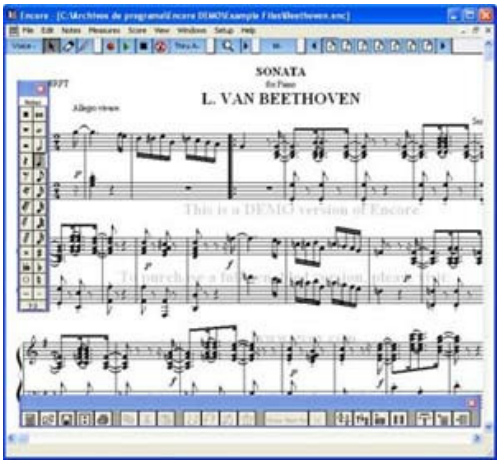

Figura 17: Ventana de Edición Principal de Encore

2.4.4.- MusicTime

MusicTime [14] es denominado como el hijo de Encore ya que sus funciones son más sencillas que éste. No obstante es capaz de crear música usando la tarjeta de sonido o una interfaz MIDI. MusicTime da la posibilidad de crear e imprimir hasta 16 pentagramas con texto. Realiza arreglos para bandas de rock, para piano y voz, jazz, coros.

Sus características principales son:

- Más de 25 plantillas de plantillas de usuario
- Limpia la pantalla de visualización.
- Modificación en tiempo real arrastrando todos los objetos.
- Auto espaciado cuando arrastra las notas, Barras y sistemas.
- Importación y exportación de archivos MIDI, también los reproduce.
- Interfaz de usuario intuitiva.
- Hasta dieciséis pentagramas con hasta ocho instrumentos
- Impresión de partituras.
- Introduzca el texto en cualquier parte de una página en cualquier tipo de letra, tamaño y estilo.
- Cambio de clave, de compás y de tempo en cualquier lugar de su puntuación.
- Imprime los archivos MIDI como partituras.
- Imprimir todo el pentagrama o páginas individuales.

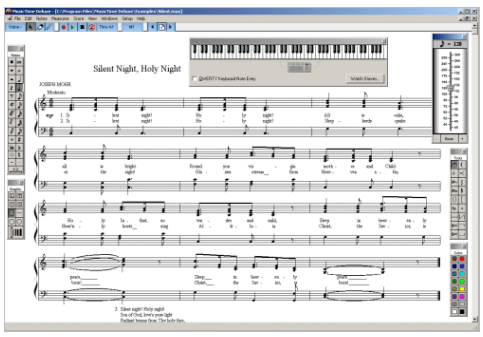

Figura 18: Ventana Principal de Edición de MusicTime

2.4.5.- Sibelius

Sibelius [27] es un avanzado programa de anotación musical que te ayudará a crear, escribir, reproducir música e imprimir tus partituras. Está diseñado para todo tipo de usuarios y músicos (estudiantes, profesores, compositores, arreglistas, etc...)

La mayor comodidad que ofrece es quizá que permite introducir las notas desde un teclado MIDI, e incluso escanear partituras e importarlas al programa.

Principalmente, el programa es capaz de:

- Diseño intuitivo con gráficos claros e instantáneos que ofrecen el mismo aspecto que las partituras manuscritas. Incluso es posible elegir el color y la textura del papel.
- Cree música con la máxima rapidez usando un teclado MIDI, el ratón o comandos de teclado.
- Puede escanear partituras,
- Agregar texto y anotaciones, copia, transporta, orquesta y crea arreglos para los instrumentos.
- Interpreta directamente las anotaciones de la partitura, incluso textos como cres. y pizz.
- Buena calidad de impresión.
- Incluye más de 60 plug-ins para agregar símbolos de acorde y digitaciones, comprobar errores, crear cuadros de escalas y arpegios, etc.
- Incluye notaciones para jazz, rock y pop (con tablatura de guitarra, símbolos de acorde y notación de batería), música antigua (con incipits, largos y bajos figurados) y música de vanguardia (con microtonalidades y ritmos complejos).

La aplicación Sibelius tendrá la siguiente forma:

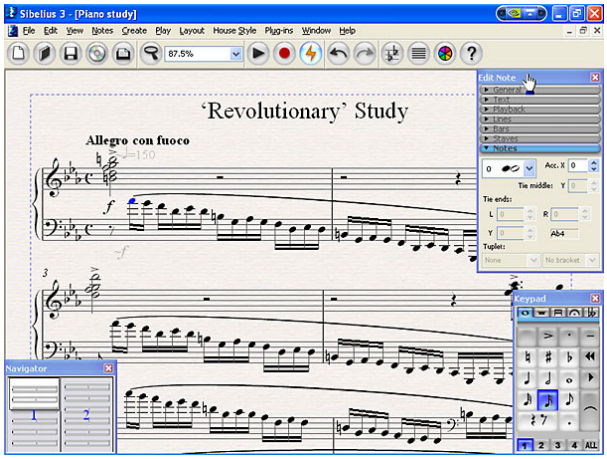

Figura 19: Ventana Principal de Edición de Sibelius

2.4.6.- Denemo

Denemo [9] es un programa de notación musical para Linux y Windows que permite introducir notación musical rápidamente a través la tipográfica LilyPond[1] para el grabado de música. La música puede ser escrita en PC (mediante el teclado), o reproducida a través de un controlador MIDI.

Denemo no da la posibilidad de imprimir partituras por si mismo sino que utiliza LilyPond, que sí es capaz de generar vistosas hojas de estilo musical en las más altas normas de publicación. Denemo se encarga de editar dichas hojas de estilo de manera eficiente.

Está organizado de manera lógica con menús y teclas o acciones del ratón que atajan dichas opciones. Dichos atajos son configurables por el usuario.

Es capaz de reproducir la música mientras el usuario se desplaza por la pantalla.

[1] LilyPond es un sistema automatizado de grabado por Han-Wen Nienhuys, Jan Nieuwenhuizen y otros. Es bonito y formatos de la música automáticamente, y tiene una sintaxis amigable para sus archivos de entrada. Es software libre ('open source').

Crea un PDF de salida con una selección de la pieza o la partitura completa. Tiene más formatos de salida como .png (para gráficos) que facilizatá su inserción en documentos o páginas web.

Sus características más notables son:

- Patrones rítmicos que ayudan en la edición musical.
- Reproducción de la música via MIDI o micrófono.
- Afinador de instrumentos musicales
- Denemo puede ser controlado por scripts.
- Comandos y atajos en los menús y el ratón son totalmente configurables por el usuario.
- Denemo permite generar una mayor variedad de tipos de música - por ejemplo, Canto Gregoriano.
- Permite la utilización de puntuación adjunta: voz, claves, compases, tempos, notas, acordes, número de traste, etc.
- Panel de vista previa a la impresión.
- El volumen y tempo sin modificables en los archivos MIDI.
- Editor de texto para poner letras: LilyPond pega el texto directamente.
- Visualización de los atributos establecidos en la partitura y por separado sobre cada movimiento: los títulos, saltos de página, etc.
- Para editarlos solo ha que hacer clic sobre ellos.
- Fácil posicionamiento del cursor con el ratón, incluso cuando hay varias voces en un pentagrama.

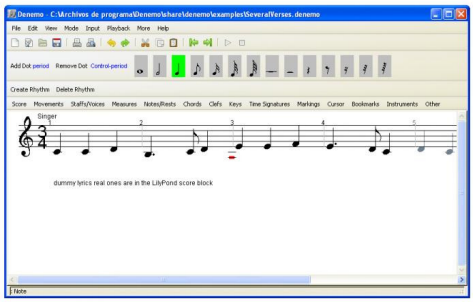

Figura 20: Interfaz del programa Denemo

2.4.7.- MuseScore

MuseScore [22] es un programa para notación musical de Linux y Microsoft Windows. MuseScore es un editor WYSIWYG, con soporte completo para reproducir partituras e importar/exportar MusicXML y archivos MIDI estándar. Tiene soporte para notación de percusión, así como impresión directa desde el programa.

El programa tiene una interfaz de usuario limpia, con una rápida entrada de notas en edición similar al ingreso rápido de notas que tienen otros programas comerciales de notación musical, Finale and Sibelius. MuseScore es Software Libre publicado bajo la Licencia Pública General de GNU [19].

A continuación se citan sus características:

- WYSIWYG (Lo que ves es lo que obtienes), las notas se escriben en una "partitura virtual".
- Número ilimitado de pentagramas y hasta cuatro voces por pentagrama.
- Inserción de notas fácil y rápida con ratón, teclado o MIDI.
- Importa y exporta MusicXml y archivos MIDI Estándar (SMF).
- Código en plataforma independiente, archivos binarios accesibles para Linux y Windows.
- Posicionamiento automático de notas en los acordes.

Figura 21: Interfaz MuseScore

2.4.8.- Otros Editores de Partituras Existentes

- **Power Tab Editor:**

Power Tab Editor [25] es una aplicación para crear música para guitarra y bajo. El programa proporciona los símbolos usados normalmente en las tablaturas.

A medida que se va creando nuestra melodía se observa como en el pentagrama de la parte superior de la aplicación van apareciendo los símbolos musicales de las notas que se van incorporando. A medida que se avanza en el desarrollo de la obra se podrá ir escuchando el resultado en el reproductor incorporado que tiene la aplicación.

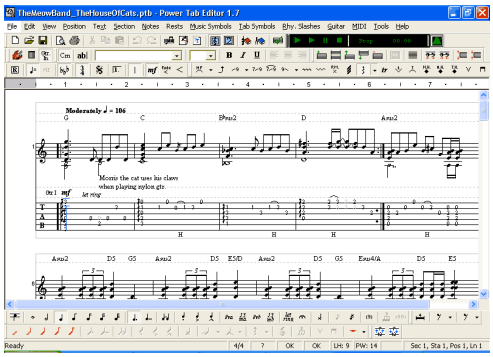

Figura 23: Interfaz Power Tab Editor

- **KBPIANO:**

Este programa [12] convertirá las teclas del ordenador en el teclado de un piano, a través del cual se podrá crear y escribir composiciones musicales de forma agradable y cómoda. También se podrán grabar en MID y reproducirlas.

Incluye, además, un metrónomo y varios simuladores de instrumentos musicales para que se complementen las melodías.

Su interfaz gráfica será la siguiente:

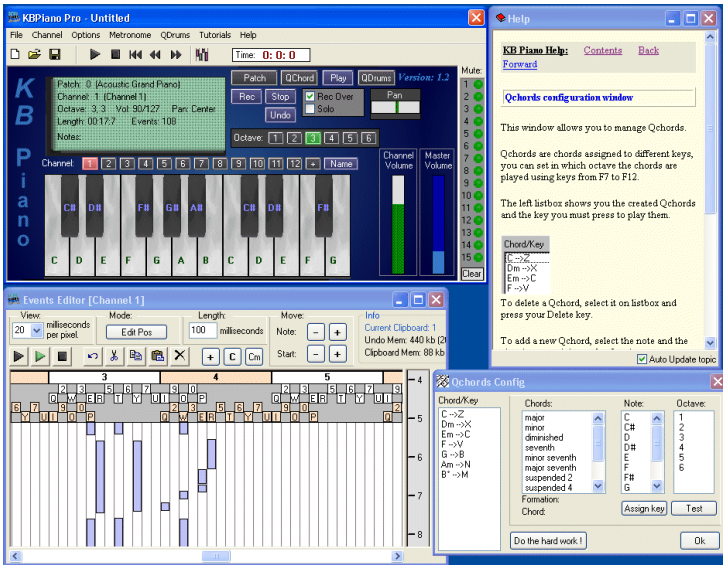

Figura 24: Interfaz KBPIANO

- **MagicScore Maestro:**

Esta herramienta [19] facilita escribir composiciones musicales. Su interfaz dispone de todos los elementos necesarios, como la conexión de dispositivos MIDI al ordenador, símbolos musicales para escribir partituras y funciones de grabado para probarlas con los instrumentos y guardarlas.

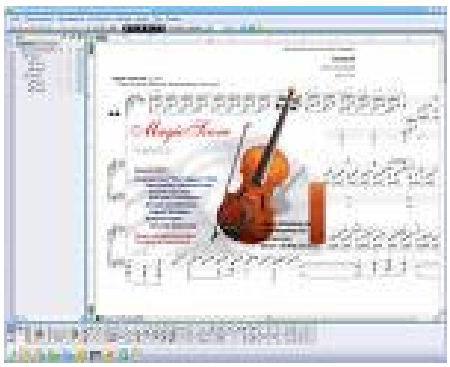

Figura 25: Interfaz MagicScore Maestro

- **NoteEdit y secuenciador MIDI:**

NoteEdit [24] es probablemente el software más completo en Linux para editar partituras con una interfaz gráfica. Debido a que las notas introducidas suenan inmediatamente en el dispositivo Midi seleccionado, NoteEdit permite incluso a principiantes crear partituras fácil y rápidamente. Con NoteEditno sólo puede fijar notas, sino también reproducir y grabar archivos MIDI. Las partituras se pueden exportar en varios formatos (entre otros MusixTeX y LilyPond).

Las notas pueden introducirse con el ratón, primero escoja la longitud y el modificador en la barra de herramientas. Se creará entonces una nota con el botón central del ratón y un espacio con el botón derecho.

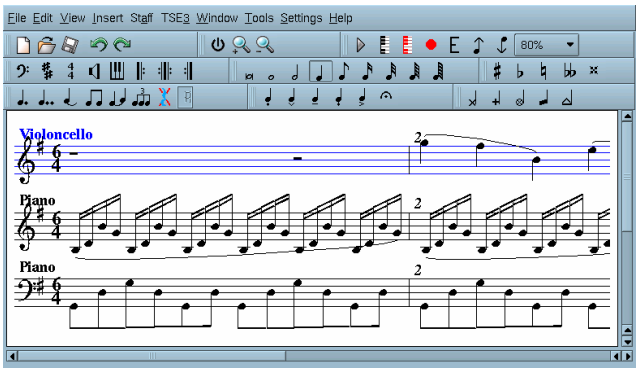

Figura 26: Interfaz NoteEdit

3.- Diseño del Sistema

3.1.- Funcionalidad del Sistema

"La música comienza donde acaba el lenguaje"
Ernst Theodor Amadeus Hoffmann

Con el objetivo de garantizar la seriedad y rigor científico del proyecto, el primer paso debe fijarse es la elaboración de un diseño previo, que incluya la delimitación de unos objetivos concretos. Esto servirá de guía de trabajo para asegurar el cumplimento de unos objetivos que sean reales y factibles, adaptados a las necesidades de un público y basándose en ciertas normas.

El objetivo principal del proyecto será, por tanto, la creación de una herramienta didáctica. El público al que irá destinada sería población que se acerca por primera vez al terreno musical y estudiantes en conservatorios u otros centros de estudio, de grado inicial.

Figura 27: Clase de música en el Colegio Público de La Navata, en Galapagar

En consonancia con dicho objetivo, el sistema debe ser una herramienta de fácil utilización, de apariencia similar a cualquier editor de textos con el que el usuario medio ya está familiarizado -esto es: una barra de menú en la parte superior, una pantalla de edición en la parte central y unas herramientas de edición con forma de botones y muy accesibles.

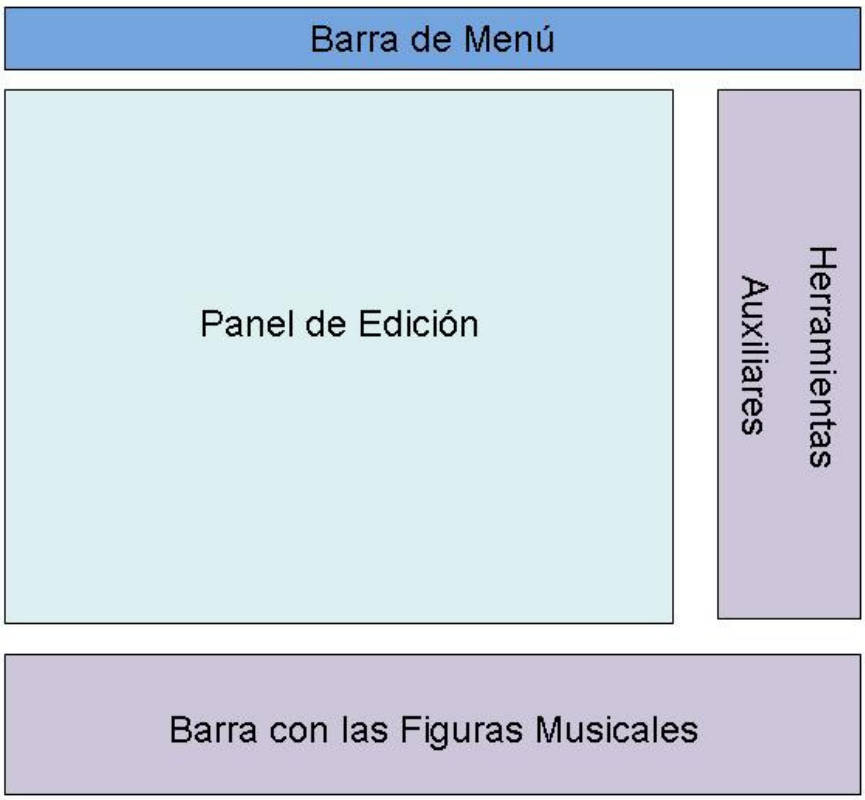

Figura 28: Esquema inicial para la interfaz gráfica

Este formato cuenta con importantes ventajas a la hora de poder lograr una función didáctica y una fácil accesibilidad al sistema:

- En primer lugar, se trata de una herramienta de edición con un diseño muy similar a otras herramientas de edición de textos o de imágenes, lo cual facilitará enormemente su uso entre estudiantes gracias a la existencia de una familiaridad previa al diseño.

- A la par, el fácil manejo de la herramienta permite dar acceso a estudiantes de colegios y de grado elemental de conservatorio, pese a que apenas hayan tenido contacto con ordenadores ni cuenten, por consiguiente, con una familiaridad previa en este terreno.

- Una tercera ventaja se logra gracias a la eliminación de la tradicional interfaz gráfica de los programas de edición en la que, para introducir las notas, se cuenta con una simulación de un piano. Este cambio resulta tremendamente importante desde un punto de vista educativo, ya que la interfaz tradicional puede confundir a los estudiantes de otros instrumentos.

- Por último, en el diseño de la interfaz se buscó que el usuario pudiera editar de manera rápida mediante el uso del ratón (por ello se situaron las herramientas a un solo clic de ratón) y de manera muy similar a la forma en la que se escriben los pentagramas sobre papel.

Sin embargo, garantizar el fácil manejo del sistema gracias al diseño de la interfaz gráfica no implica necesariamente la consecución de una herramienta didáctica. Es por ello que el proyecto se plantea unas metas más ambiciosas en lo que se refiere a métodos de educación musical.

Puesto que es una herramienta didáctica, es necesario garantizar la cobertura de los contenidos y conceptos musicales que deben adquirirse en los estudiantes que se encuentran en el nivel de aprendizaje al que va dirigido el sistema.

Por lo tanto:

- En primer lugar, se ha diseñado un instrumento capaz de cubrir las necesidades básicas de los estudiantes de esa edad con relación a notaciones musicales.

Dichas necesidades o conceptos básicos que se desean cubrir son:

- Colocación de las notas en la clave de Sol.
- Identificación de los distintos compases básicos (dos por cuatro, tres por cuatro y cuatro por cuatro).
- Relación entre la duración de las distintas figuras.
- Aprender a colocar los silencios y relacionarlos en función de sus duraciones.
- Otras notaciones musicales: sostenidos y bemoles.

- En segundo lugar, el sistema permite que los alumnos puedan relacionar cada nota con su sonido (frecuencia) ya que tiene la posibilidad de reproducir las melodías.

- El sistema es capaz de grabar la melodía en un archivo MIDI y recuperarla más adelante.

- Dado su carácter de Editor, las notas podrán ser borradas y también se podrá borrar la última línea del pentagrama.

- La aplicación es capaz de llevar la cuenta de la duración de las figuras que se van introduciendo. De esta forma, cuando se rellena un compás la aplicación lo detecta y pinta automáticamente una línea de separación entre los compases.

- A la par, facilita que los estudiantes aprendan a diferenciar entre los distintos instrumentos ya que se pueden generar archivos musicales MIDI con instrumentos de viento (clarinete y flauta), de cuerda (guitarra y violín), percusión (xilófono) o cuerda percutida (piano).

Figura 29: Instrumentación de una orquesta

- El sistema permite, también, aprender los distintos tempos en los que se puede ejecutar una partitura ya que, a parte de seleccionar el tipo de instrumento con el que se desea reproducir, se puede seleccionar el tempo: rápido, medio o lento.

- La aplicación será capaz de llamar a módulos externos complementarios que hagan del sistema una potente y completa herramienta didáctica.

    - De manera sencilla podrá ser capaz de cargar un módulo externo generador de dictados musicales.

    - Así mismo, podrá ser capaz de cargar un módulo armonizador de melodías que sirva de herramienta para estudiar el encadenamiento de diversas notas superpuestas: es decir, la organización de los acordes[2].

- Por último, la herramienta permite acercar a estos estudiantes a las nuevas tecnologías. Y puesto que Internet es un medio para compartir datos, se busca que el programa devuelva pentagramas en formato web (MusicXML). Esto hará del sistema una herramienta basada en estándares.

[2] Se llama "acorde" a la combinación de tres o más notas diferentes que suenan simultáneamente

3.2.- Introducción al Diseño: "Diseño Interactivo"

"En la música todos los sentimientos vuelven a su estado puro y el mundo no es sino música hecha realidad."

Arthur Schopenhauer

Toda aplicación debe seguir unos principios básicos que garanticen una construcción óptima de la misma [26]. Es por ello que la elaboración de dichos principios ha constituido una de las principales prioridades del diseño previo de la interfaz gráfica de este proyecto. No en vano, dicho diseño sienta las bases sobre las que se empezó a construir esta aplicación multimedia.

Se exponen a continuación los principales principios-guía del proyecto junto con una previsión de cumplimiento de los mismos.

- *Principio de la múltiple entrada*

En el almacenamiento de la información intervienen tres parámetros: el cognitivo, el afectivo y el factor de la experiencia previa. Todos ellos deben ser tenidos en cuenta en la interfaz gráfica.

Por lo que respecta a los parámetros cognitivos, éstos hacen referencia dentro del campo de la psicología a la forma en que nuestro cerebro procesa la información. Aplicados a la música, los parámetros cognitivos se inscriben dentro de la esfera de la notación musical, En ella, la única referencia con la que se cuenta son los medios tradicionales en los que los músicos escribían sus piezas: los pentagramas en papel.

Dichos pentagramas, no sólo cubren el parámetro cognitivo, sino también el de la experiencia previa (puesto que los pentagramas en papel son un medio conocido por cualquier aficionado a la música, al menos desde un punto de vista representativo).

Por tanto, el diseño de la interfaz gráfica se basará en gran medida en los formatos tradicionales.

- *Principio de interactividad*

Deberá planificarse cuidadosamente cada interacción, como si fuera una tarea diferenciada, con el objetivo de lograr un funcionamiento y aplicación integral de la herramienta.

Para ello, resulta imprescindible identificar, en primer lugar, el conjunto de posibles interacciones que abarcan el desarrollo completo de la aplicación. En concreto, seleccionar entre las posibles duraciones y emplazar las notas en el pentagrama se han establecido como dos de las interacciones prioritarias.

- *Principio de necesidad*

La aplicación debe ser útil (necesidad de la existencia de la aplicación) y multimedia (necesidad de ser diseñada bajo ese enfoque). Se intentará cumplir este principio: al diseño le corresponde garantizar la parte multimedia.

- *Principio de atención*

Uno de los objetivos principales de la aplicación es mantener la atención sostenida: para ello resulta necesario apelar a todos los tipos de atención presentes en cualquier persona (cognitiva y afectiva, principalmente).

La atención cognitiva se basa en el valor de la información suministrada, por lo que la aplicación debe garantizar la aportación de información relevante, bien organizada. Por su parte, la atención afectiva se basa en el lazo afectivo que se establece entre el usuario y la aplicación.

3.3.- Diagrama General del Sistema

"El arte de la música es el que más cercano se halla de las lágrimas y los recuerdos"

Oscar Wilde.

Una vez fijados los objetivos y después de haber visto en qué se basa el diseño de la aplicación, se presenta el diagrama general del sistema.

Cabe destacar que para facilitar la comprensión entre los distintos módulos se va a diferenciar el diseño en dos niveles:

- Diseño a nivel gráfico: cómo se representan las notas, justificación del cumplimiento de los objetivos en el diseño de la interfaz.

- Diseño a nivel estructural de los datos: cómo se consigue la conversión multiformato, cómo se representa la información que se necesita compartir entre ambos módulos y justificación del cumplimiento de los objetivos y las tecnologías usadas.

Como es evidente, ambas partes están relacionadas así que, inicialmente, se mostrará un diagrama con todos los bloques que describen el sistema y posteriormente se desglosará el contenido de los mismos.

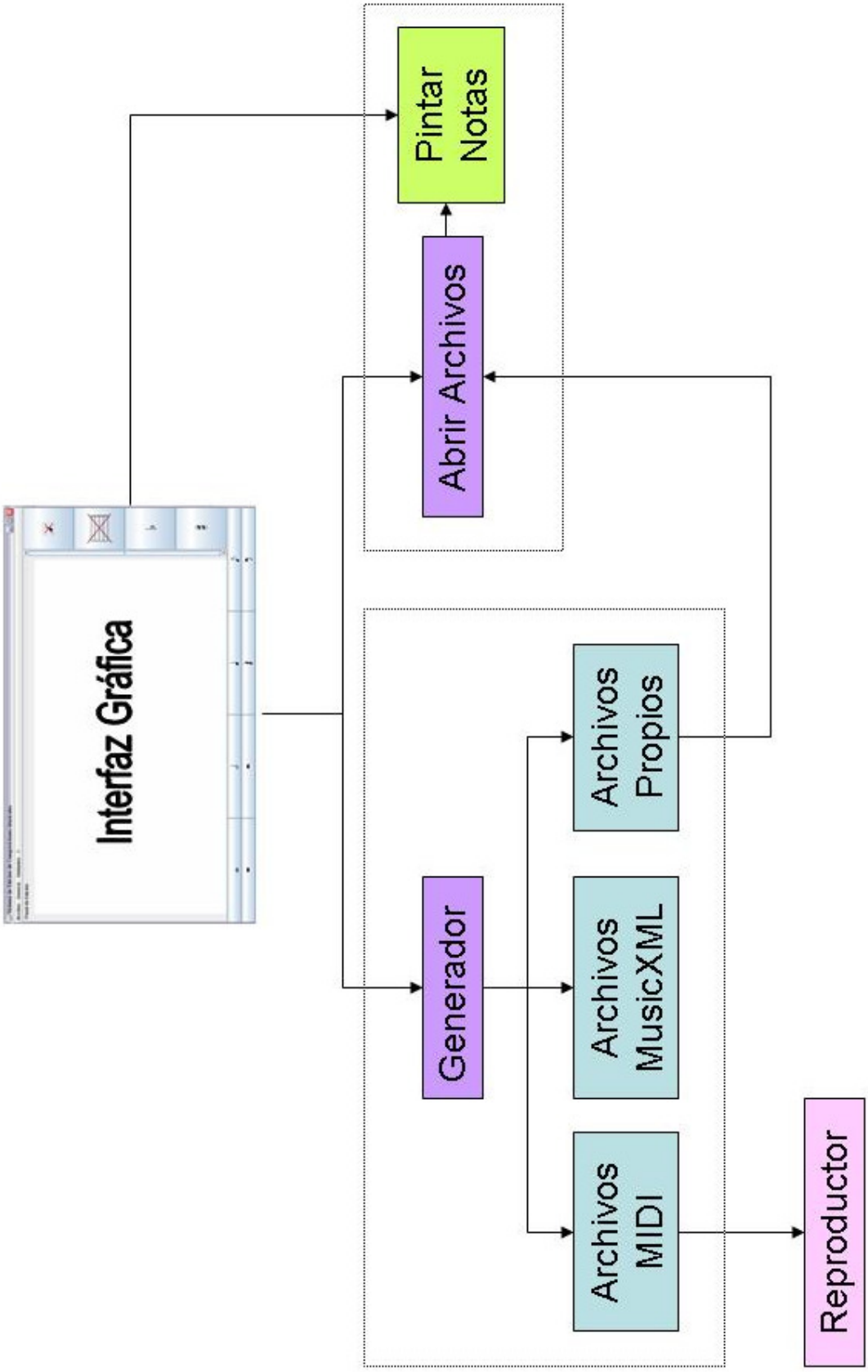

Figura 30: Diagrama de Bloques del Diseño del Sistema

La interfaz será, en todo momento, el eje central de la aplicación ya que servirá de método para lograr la interactividad entre el usuario y el ordenador.

Desde ahí se encontrarán las dos funcionalidades distintas del sistema. La primera de ellas, en relación con la estructura de datos, será la encargada de generar tres tipos de archivos: Archivos propios (que podrán ser pintados de nuevo para su edición), archivos MIDI (que podrán ser reproducidos gracias a un reproductor interno) y archivos MusicXML.

La segunda de las funcionalidades del sistema tiene que ver con la representación gráfica y el diseño de la interfaz. Este módulo será por tanto el módulo gráfico y se encargará principalmente del pintado de notas.

3.4.- Módulos Gráficos

"El genio tiene que estar siempre en lucha entre las fuertes intuiciones que siente dentro como un volcán y la limitación de sus medios de expresión. Es una lucha interior. El resultado siempre es doloroso y por eso la música no puede ser nunca alegre"

Giovanni Nenna

Para describir los módulos gráficos, primero se explicará cómo se ha diseñado, en líneas generales, la interfaz y posteriormente el método que pinta las notas.

Es importante comprender cómo se ha ordenado la interfaz y familiarizarse con la situación de los comandos más utilizados para, posteriormente, poder pintar las notas con destreza.

3.4.1.- Creación de la Interfaz Gráfica

El diseño de la interfaz gráfica, como se dijo en el capítulo anterior, se basa en una serie de principios que se han respetado en este proyecto: el principio de múltiple entrada, de interactividad, de necesidad y de atención.

Se busca una interfaz fácil de utilizar, accesible a todos los públicos, cuyo funcionamiento sea similar al de un pentagrama tradicional y que cumpla unos conceptos musicales básicos.

Se podría dividir la interfaz gráfica en cuatro bloques distintos: Barra de menú, panel de figuras, panel de herramientas adicionales y panel de edición.

- Barra de Menú: ahí se encuentran cuatro submenús: archivo, generar, unidades y ayuda.

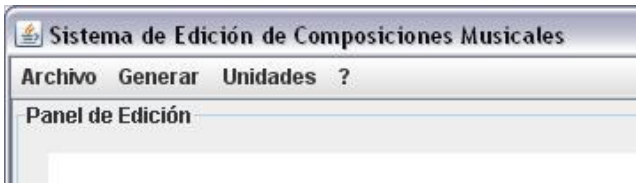

Figura 31: Barra de Menú

- Panel de Figuras: se han utilizado duraciones de redonda, blanca, negra y corchea con sus correspondientes silencios. Están colocadas en dos filas de botones

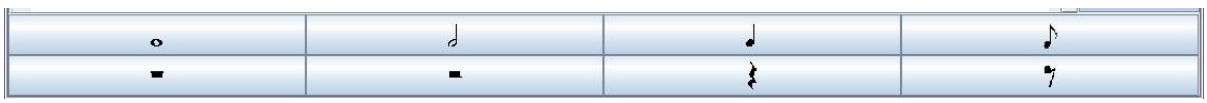

Figura 32: Panel de Botones

Cuando se selecciona alguno de los botones y posteriormente sobre el panel de edición, la aplicación pintará la figura o silencio seleccionada anteriormente en el emplazamiento elegido.

- Panel de herramientas auxiliares: aquí se encuentra otro panel con cuatro botones. Ahí se encontrarán las opciones de borrar la última nota, borrar la última línea de pentagrama y aplicarle a la nota que se va a pintar una modificación tonal de medio tono superior (sostenido) o inferior (bemol).

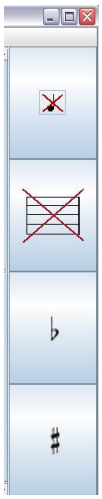

Figura 33: Panel de Herramientas Auxiliares

- Panel de edición de partituras: en este panel se encuentra una distribución de imágenes por capas para poder superponerlas y un scroll para ampliarla en caso de necesidad.

  Al inicializar la aplicación se encontrará simplemente una capa en el nivel inferior de una imagen blanca. Cuando se agregue un pentagrama (archivo> nuevo> 3 /4), se incluirá en la siguiente capa un pentagrama y en la siguiente una clave de Sol y la grafía del compás.

Figura 34: Panel de edición

El resultado que se obtendrá después de esta descripción será el conjunto de la interfaz gráfica, mostrado en la siguiente figura.

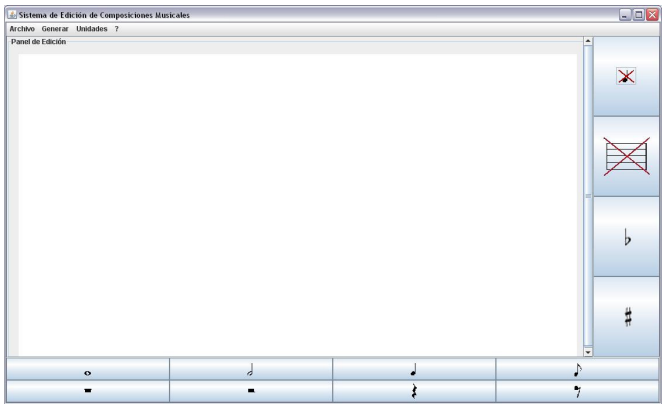

Figura 35: Interfaz Gráfica

3.4.2.- Pintado de Notas en el Panel de Edición

El pintado de notas es una de las partes fundamentales de este proyecto. Se realiza según el siguiente diagrama de flujo:

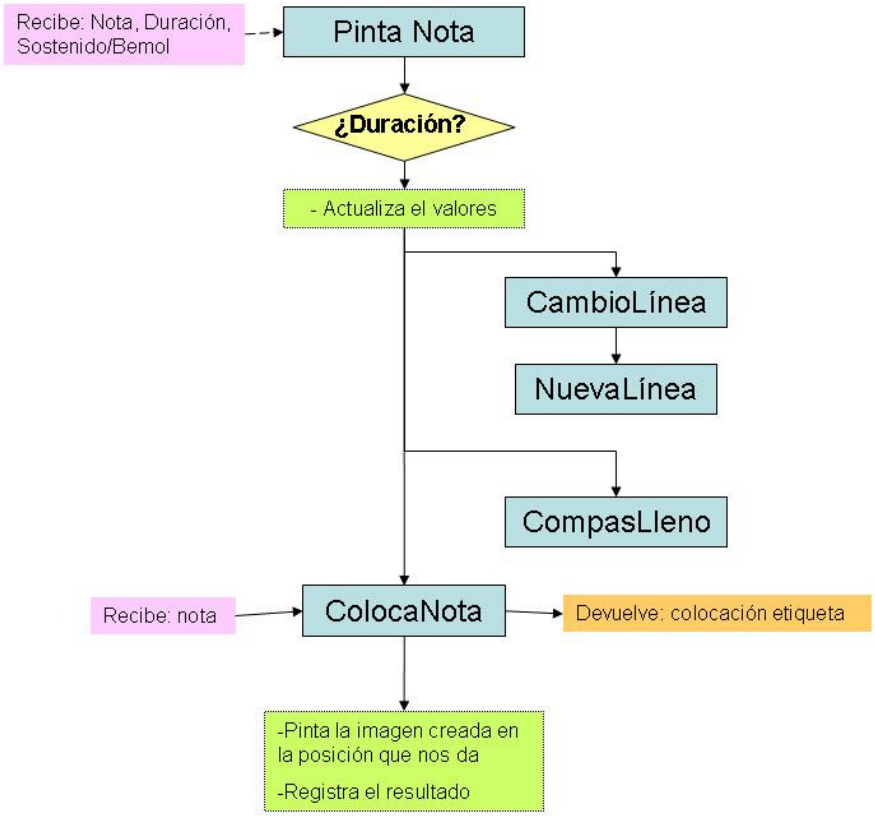

Figura 36: Método pintaNota

El método que pinta la nota se basa en la duración de la figura que se seleccione (en los botones de la barra inferior: redonda, blanca, negra o corchea) y el lugar del pentagrama en el que lo se hará (Do, Re, Mi, Fa, Sol, La o Si).

El resultado, como es de esperar, es la figura seleccionada pintada en el lugar del pentagrama que se desea. No obstante se requiere de ciertas comprobaciones antes de pintarla.

Primero, el programa elige con qué figura se está trabajando y, en función de dicha figura comprobará:

- SI la línea de pentagrama en la que va a pintarla está llena o puede seguir pintando ahí. Si estuviera llena dibujará una nueva línea y si no continuará.

- Si el compás en el que se está dibujando está lleno o no. Es decir, si se trabaja desde un compás de tres por cuatro y se ha dibujado una negra y una blanca (el compás ya estará lleno), antes de dibujar la siguiente nota el programa dibujará una línea de separación de compases y empezará la cuenta de nuevo.

- Comprobará también dónde se ha hecho clic y devolverá el lugar en el que se debe dibujar la figura.

Posteriormente se tendrá el resultado esperado: una figura en el emplazamiento adecuado y su correspondiente registro.

3.5.- Módulos de Estructura de Datos

"Error funesto es decir que hay que comprender la música para gozar de ella. La música no se hace, ni debe jamás hacerse para que se comprenda, sino para que se sienta."

Manuel de Falla

En el apartado anterior se ha comentado en varios momentos que los datos se almacenaban. Ahora se va a explicar en qué consiste dicho almacenamiento y para qué sirve.

Antes de empezar, es importante recordar que sólo hay un momento en el que se almacenen las notas y es justo después de ser pintadas. Toda nota que no se represente gráficamente no será almacenada.

3.5.1.- Almacenamiento de Datos

Cada nota cuenta con unas cualidades propias. Esto es lo realmente importante a la hora de almacenar cada nota. Se deben guardar, por tanto, los siguientes parámetros: nota, duración, lugar en el que se está pintando, sostenidos y bemoles.

Los archivos serán guardados en varios formatos anteriormente explicados: archivos propios (que darán la posiblidad de volverlos a pintar y modificarlos), archivos MusicXML y archivos MIDI (que podrán ser reproducidos).

Los datos serán almacenados en el momento en el que se pintan y accederán a ellos los módulos que se muestran en la siguiente figura:

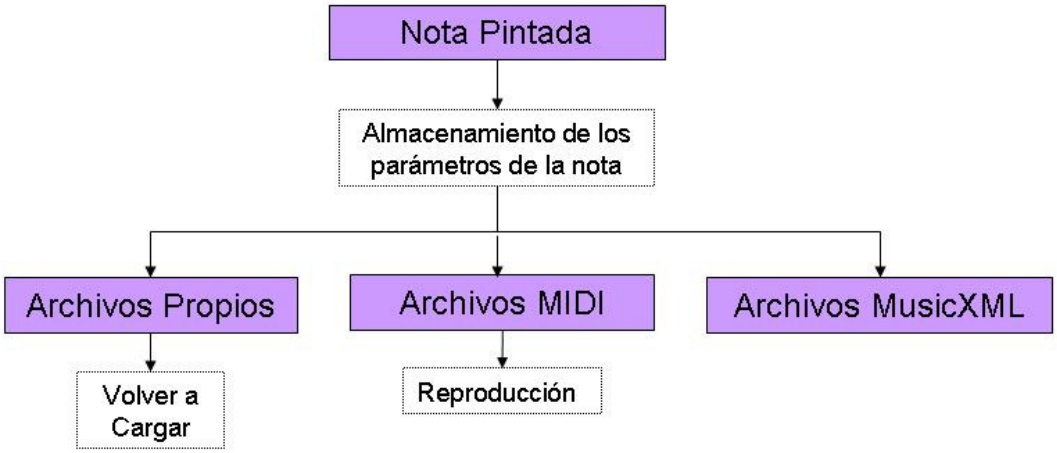

Figura 37: Archivos Generados

3.5.2.- Cargar y Pintar pentagramas

En el caso de archivos que se leen deben estar en formato .mp, un formato de archivos propio de la edición del programa. Esta acción se realizará al escuchar el evento del botón del menú “Abrir”.

Los archivos tendrán una estructura propia que facilitará a la aplicación el proceso de pintado de los pentagramas que hayan sido guardados previamente. Dicha estructura se explicará más adelante en el capítulo de implementación.

Como en los casos anteriores se va a explicar la relación entre los métodos para determinar su dependencia.

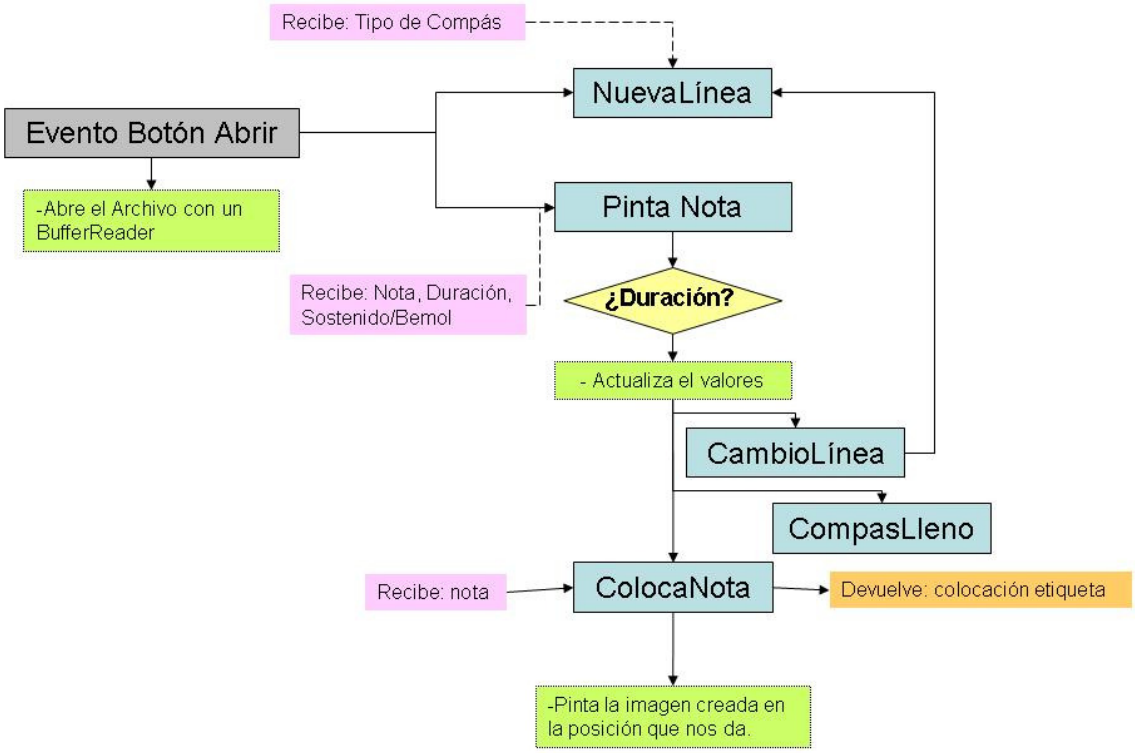

Figura 38: Evento Abrir

El comportamiento en este caso es similar al de escribir una nota por primera vez. Desde el botón Abrir se genera un evento que activará la lectura de datos y pintará una nueva línea con el compás adecuado (con el que se guardó el archivo).

Posteriormente irá leyendo una a una todas las líneas, en cada línea habrá la información necesaria para pintar cada nota: duración, nota, parámetros necesarios para saber en qué línea estaba guardada y por último, como en el caso anterior, se pintará la nota en la colocación adecuada.

En este caso no se almacenará la información de las notas repintadas. Tras abrir el archivo se podrá continuar pintando las notas y en este caso sí se almacenarán las nuevas notas a continuación de las anteriores.

También se podrán borrar las notas del archivo utilizando los controles en la barra de herramientas lateral: borrar nota, borrar línea; y por tanto, se podrán sobrescribir.

3.5.3.- Guardar Archivos MusicXML

El almacenamiento de los datos consiste en ir guardando cada nota en un archivo. Para el caso anterior las normas que dictan la forma de guardar cada nota es elegida por los objetivos del proyecto. Para este caso, las normas de almacenamiento se han basado en las normas de MusicXML. Es decir, se aplicará un estándar a la hora de guardar los datos.

3.5.4.- Guardar Archivos MIDI

Con este apartado se termina el último bloque del diseño del sistema. Para generar los archivos MIDI, como en ambos casos anteriores, se recurrirá a los datos guardados. La principal diferencia es la forma en la que se generan los archivos ya que no se utiliza una escritura normal sino una escritura de archivos MIDI.

Al final, se generará un archivo MIDI al que se podrá acceder posteriormente para su reproducción. Los archivos se podrán generar con distintas opciones: seis tipos de instrumentos y tres tipos de tempos para cada instrumento con el siguiente orden:

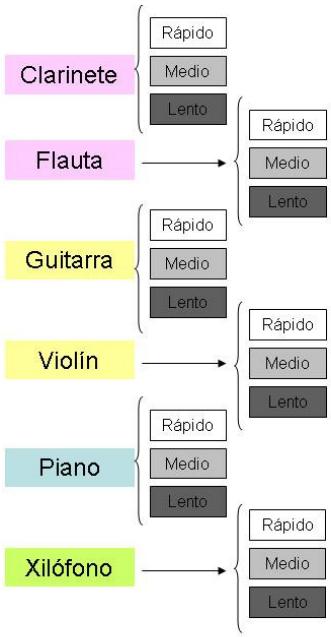

Figura 39: Distintas opciones de archivos MIDI

4.- Implementación

En el capítulo de implementación se explicará paso a paso la creación de la aplicación. Para facilitar su comprensión se dividirá el capítulo en dos: implementación de los módulos gráficos e implementación de los módulos generadores (es decir, analizar la estructura de datos de las clases que generan y reproducen los archivos).

4.1.- Diagrama de Clases

"Es imposible traducir la poesía. ¿Acaso se puede traducir la música?".

François-Marie Arouet

Para entender bien la estructura que sigue la aplicación se mostrará, en la siguiente figura, un diagrama de clases y se explicará la función de cada una de ellas:

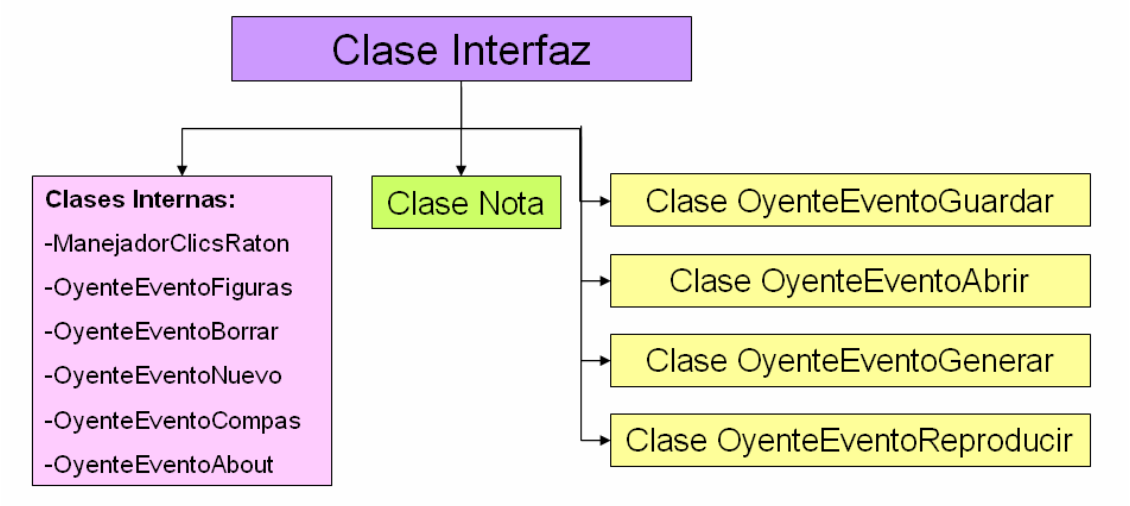

Figura 40: Diagrama de Clases

Como se puede apreciar, se han dividido las clases en tres grupos principales. El primero de ellos se encarga de escuchar los eventos relativos a las interfaces gráficas tales como los que se generan en el panel de edición, al crear archivos nuevos, al borrar figuras, al pulsar botones de figuras o modificaciones tonales, etc. Son internos a la clase *Interfaz*.

El segundo bloque solo contiene la clase *Nota* que, como se explicará más adelante, guarda información relativa a cada nota que se dibuja. Es una clase de apoyo.

El último bloque, a cambio, es el que se ocupa de escuchar eventos relativos a la creación de archivos, esto es: guardar y abrir ficheros, generar archivos MIDI y posteriormente reproducirlos. Todas estas clases son externas a la clase *Interfaz* pero se comunican con ésta para conseguir los datos de las notas ya pintadas.

Para aclarar aún más estos conceptos, se va a mostrar sobre la interfaz qué clase escucha cada componente gráfico.

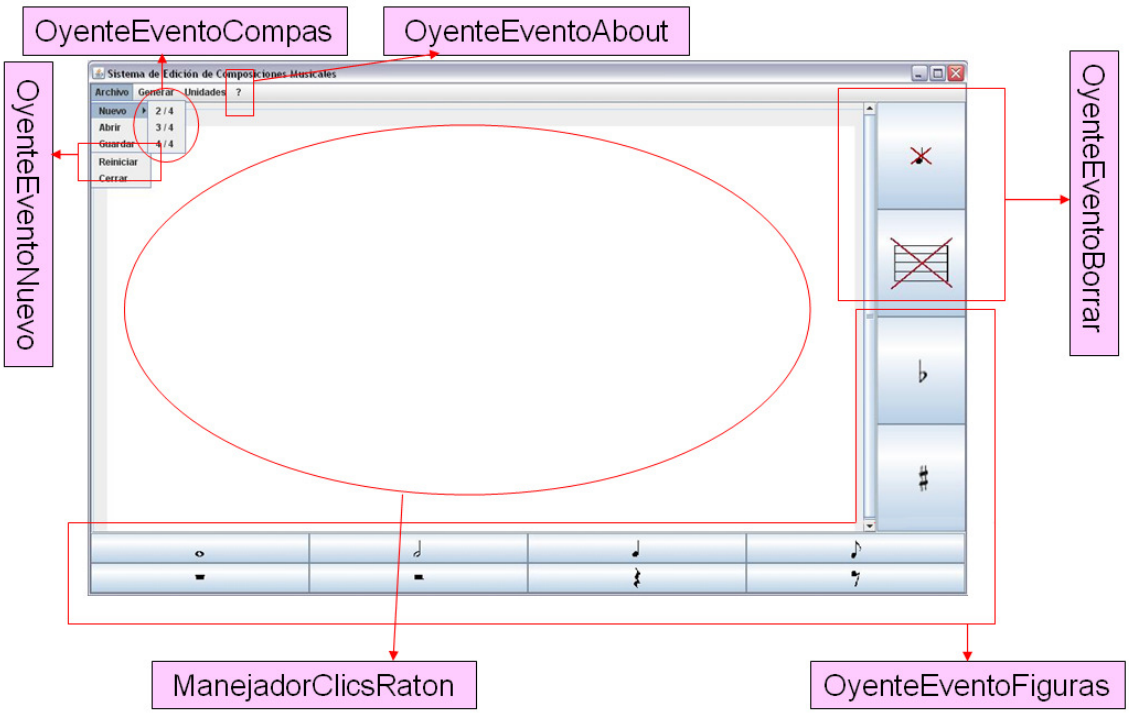

Figura 41: Eventos que escuchan las clases internas

Como se puede comprobar, las clases internas son las encargadas de escuchar los eventos que tienen relación con la creación de nuevos pentagramas, su reinicio, los botones de figuras, de borrado, de modificaciones tonales, el panel de edición y el submenú de ayuda.

A cambio, las clases externas se encargarán de implementar los eventos mostrados en la siguiente imagen.

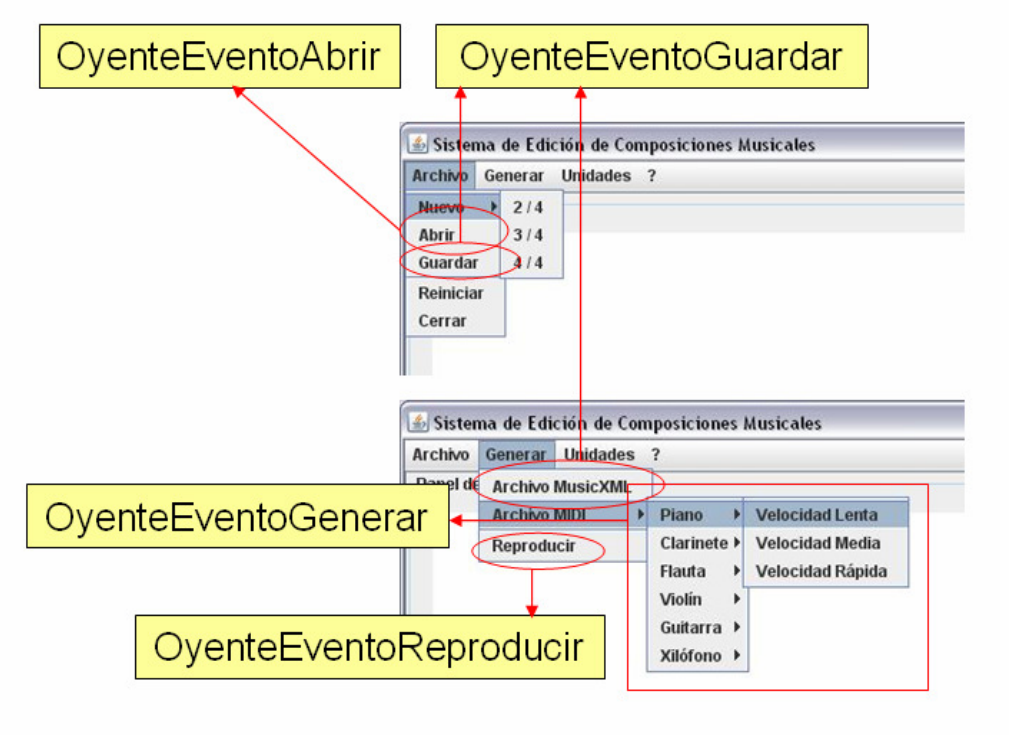

Figura 42: Eventos que escuchan las clases externas

Se observa claramente la división en dos grupos: los que se encargan de los módulos gráficos y los que se encargan de la creación, modificación y reproducción de archivos.

4.2.- Implementación de los Módulos Gráficos

"Dadme el mejor piano de Europa, pero con un auditorio que no quiere o no siente conmigo lo que ejecuto, y perderé todo el gusto por la ejecución."

Wolfgang Amadeus Mozart

En el capítulo anterior se explicó la forma que se quería que tuviera la interfaz gráfica y se justificó el porqué. Antes de comenzar en este apartado con la implementación, se debe aclarar que todas las acciones del módulo gráfico se desarrollarán en la clase principal: *Interfaz.*

4.2.1.- Creación de la Interfaz Gráfica

Para realizar la creación de la interfaz gráfica se debe prestar especial atención a cada uno de sus componentes. Cada componente debe ser declarado, creado y añadido a otro componente o a un contenedor.

Los contenedores son tipos especiales de componentes que están formados por uno o más componentes (o por otros contenedores). Eso sí, todos los componentes agrupados en un contenedor se pueden tratar como una sola entidad.

En el caso de la aplicación existe un contenedor principal, *JFrame*, que proporciona una ventana principal de aplicación. Se contará con otros contenedores: *JMenuBar* (para la barra de menú), *JPanel* (para los botones con notas) y *JScrollPane* que lleva añadido un *JLayeredPane* (la primera da la opción de tener un scroll, y la segunda componentes por capas).

Los componentes que se necesitan son de distintos tipos, los más utilizados son: *JLabel* (una etiqueta que llevará una imagen tipo *ImageIcon* añadida), *JButton* y *JMenuItems*.

La organización de la interfaz en función de sus componentes y contenedores será la mostrada en la figura siguiente:

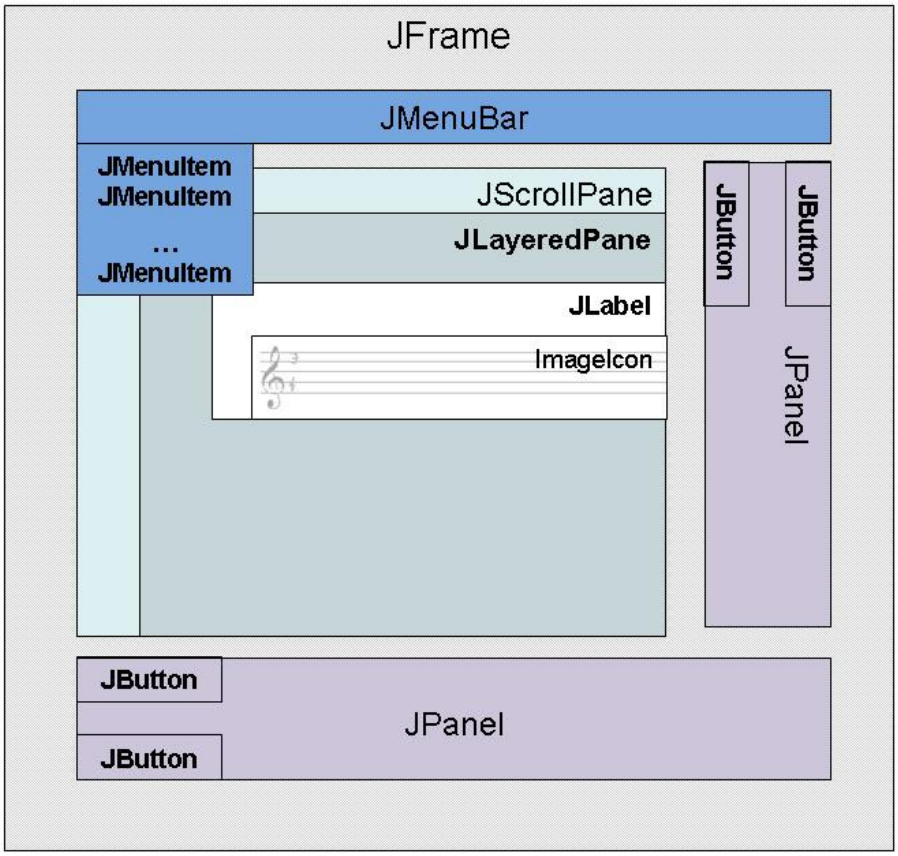

Figura 43: Componentes y Contenedores de la Interfaz

Lo que realmente hará que la aplicación funcione son los escuchadores de los posibles eventos que generan los componentes. Es decir, el objeto fuente registra qué objetos oyentes están interesados en recibir algún evento, cuando sucede el evento (por ejemplo, hacer clic en un botón), el objeto fuente se lo comunica a todos los oyentes registrados.

En este caso, se entiende como evento un objeto que encapsula toda la información sobre una interactuación del usuario con la interfaz gráfica. Los eventos, que son recogidos por la aplicación, son *ActionListener* para todos los *JButton* y *JMenuItem* y, para el caso del panel de edición, un *MouseListener* que devolverá la posición en la que se hace clic.

De esta forma, cuando se inicial la aplicación, el proceso que sigue la misma hasta encontrarse preparada para funcionar será el mostrado en la figura siguiente:

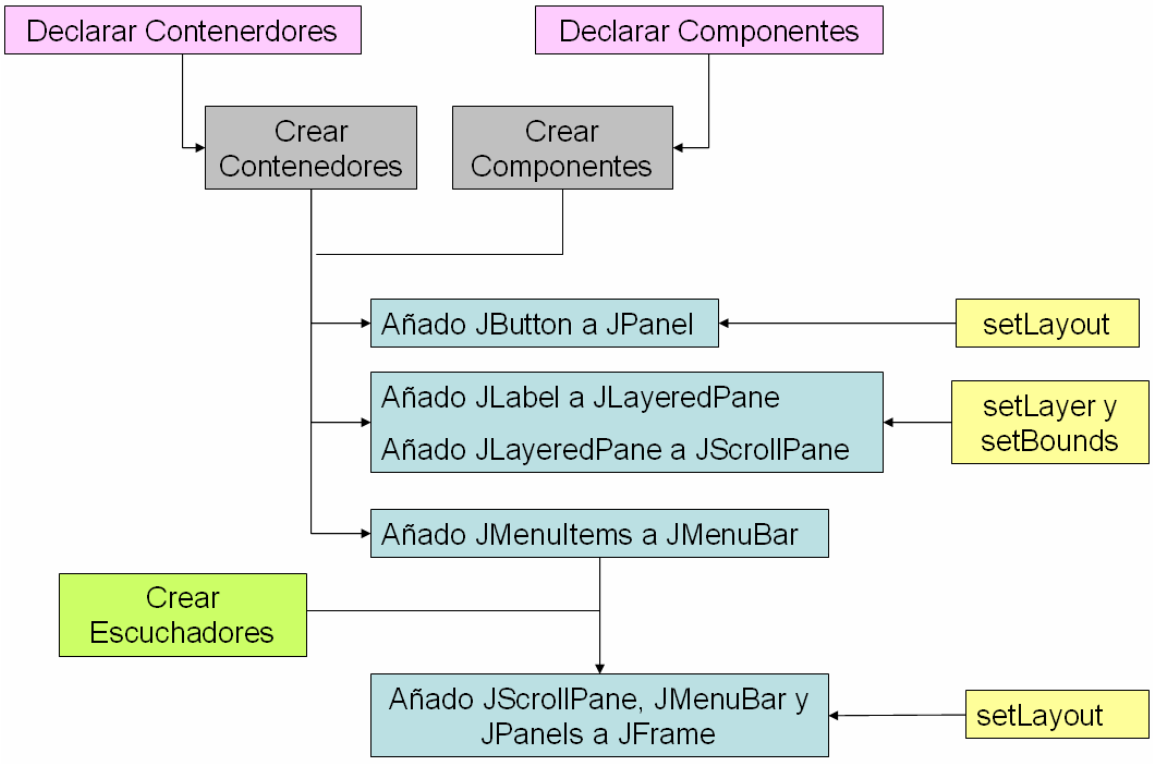

Figura 44: Creación de la Interfaz

Se declaran y se crean tanto los contenedores como los componentes. Se añaden a cada contenedor su componente (en el caso de los paneles con botones) y se configuran mediante *Layouts*.[3]

[3] Los Layouts ayudan a adaptar los diversos componentes que se desean incorporar a un panel, es decir, especifican la apariencia que tendrán los componentes a la hora de colocarlos sobre un contenedor.

Cabe destacar que en este caso el *Layout* utilizado es el *GridLayout*, que organiza los componentes en rectángulos regulares, de 4x2 en el panel inferior y de 1x4 en el panel de la derecha.

En algunos casos se añadirá a cada contenedor uno o varios componentes y el resultado será añadido a otro contenedor que dé nuevas opciones. Este es el caso del panel de edición en donde hay un *JScrollPane,* que da la opción de poder seguir añadiendo líneas de pentagrama, y dentro de éste habrá un *JLayeredPane*, gracias al cual se pintan imágenes unas encima de otras por capas.

Según se ha diseñado, en la capa inferior (*DEFAULT_LAYER*) se encontrarán los pentagramas, después (*PALETTE_LAYER*) los compases y las claves, y más arriba (*MODAL_LAYER*) las figuras. Guardando aún dos profundidades (*POPUP_LAYER* y *DRAG_LAYER*) para las modificaciones tonales y posibles ampliaciones que posteriormente se apliquen a la aplicación.

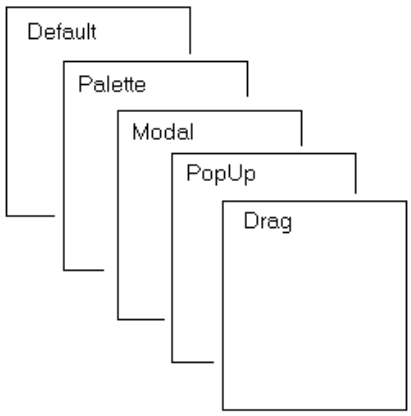

Figura 45: Distribución de Capas en JLayeredPane

Un caso más sencillo es los de los *JMenuItems* que carecen de configuraciones, simplemente se deberán añadir a la barra de menú (*JMenuBar*) en el orden que se desee que aparezcan.

El paso final para la creación de la interfaz sería añadir todos los contenedores al contenedor principal, es decir, al contenedor *JFrame*. No obstante, no es ese el proceso ya que, antes de mostrar una configuración definitiva, se le añaden los escuchadores a los componentes y, cuando ya estén activos, se añadirán al *JFrame* principal y empezará su actividad.

También se debe configurar el *Layout* con el que se dispondrán los distintos contenedores dentro del *JFrame*. Para este caso, el *Layout* seleccionado es *BorderLayout*, que organiza los contenedores en función de los puntos cardinales: Norte (barra de menú), sur (panel de figuras), este (panel auxiliar), centro (panel de edición) y oeste (no utilizado).

Tras saber la relación entre los componentes y contenedores y haber explicado su funcionamiento, en cuanto a los eventos se refiere, se explicarán las distintas opciones que tiene el sistema: cómo crear el primer pentagrama, cómo se pintan las notas y cómo se borran.

4.2.2- Dibujar el Primer Pentagrama

Esta clase se inicializará al escuchar los eventos generados de hacer clic en alguno de los tres tipos de compás en Archivo>Nuevo. Es importante aclarar que esta clase interna escucha tres eventos distintos: compás de dos por cuatro, de tres por cuatro y de cuatro por cuatro.

La clase interna *OyenteEventoCompas* implementará *ActionListener* y capturará el evento *ActionEvent*. Si el evento es generado por la opción dos por cuatro, llamará al método *nuevaLinea()* dándole como atributo el valor de la variable de tipo *int*, *TipoCompas,* y desactivará los *JMenuItems* que están añadidos en nuevo para que, en mitad de la edición de partituras, el usuario no pueda cambiar el compás.

Se realizará el mismo procedimiento para el resto de posibles compases, salvo que el valor de la variable de tipo *int* será distinta para cada caso.

El método *nuevaLinea()* es el encargado de pintar cualquiera de las líneas del pentagrama. Se hará referencia a él en dos momentos distintos: cuando se pinta la primera línea y cuando la línea en la que se está editando está completa. A continuación se muestra el método *nuevaLinea().*

```
   public void nuevaLinea (int TipoCompas){

     if (TipoCompas==24){//Variable int que da el tipo de compás
        JLabel eCompas2=crearFigura(new ImageIcon("24.gif"));//Se crea
la figura con la imagen correspondiente
        eCompas2.setBounds(60,133+(ipentagrama.getIconHeight()*linea),
icompas2.getIconWidth(), icompas2.getIconHeight());//Se coloca
        mPentagramas.add(eCompas2, JLayeredPane.MODAL_LAYER);//Y se
dibuja
     }
     else if (TipoCompas==34){//Idem compas tres por cuatro
        JLabel eCompas3=crearFigura(new ImageIcon ("34.gif"));
        eCompas3.setBounds(60,133+(ipentagrama.getIconHeight()*linea),
icompas3.getIconWidth(), icompas3.getIconHeight());
        mPentagramas.add(eCompas3, JLayeredPane.MODAL_LAYER);
     }
     else if (TipoCompas==44){//Idem compas cuatro por cuatro
        JLabel eCompas4=crearFigura(new ImageIcon ("44.gif"));
        eCompas4.setBounds(60,133+(ipentagrama.getIconHeight()*linea),
icompas4.getIconWidth(), icompas4.getIconHeight());
        mPentagramas.add(eCompas4, JLayeredPane.MODAL_LAYER);
     }

     JLabel ePentagrama=crearFigura(new ImageIcon
("pentagrama.gif"));//Se crea figura pentagrama
     JLabel eClave=crearFigura(new ImageIcon ("sol.gif"));//Idem Clave
   //Se situa pentagrama, clave y se dibujan
     ePentagrama.setBounds(20,90+(ipentagrama.getIconHeight()*linea),
ipentagrama.getIconWidth(), ipentagrama.getIconHeight());
     eClave.setBounds(35,130+(ipentagrama.getIconHeight()*linea),
iclave.getIconWidth(), iclave.getIconHeight());
     mPentagramas.add(ePentagrama, JLayeredPane.PALETTE_LAYER);
     mPentagramas.add(eClave, JLayeredPane.MODAL_LAYER);
     linea++;//Se aumenta el valor de la línea
     mScroll.revalidate();//Ampliamos el scroll
  }
```

Figura 46: Método nuevaLinea()

Su funcionamiento es el siguiente: en función del tipo de compás creará una etiqueta *JLabel* con la imagen *ImageIcon* adecuada (para realizar esta acción recurrirá al método *crearFigura (ImageIcon)* que devuelve la etiqueta), pintará dicha etiqueta con el método *setBounds()* en un lugar determinado del panel de edición que estará relacionado con la línea en la que se desea pintar. Se añade la etiqueta al panel contenedor.

Siguiendo este criterio se crearán etiquetas del pentagrama y la clave (con el método *crearFigura (ImageIcon)* usado anteriormente), se sitúan en el lugar correcto, se añaden al panel y se revalida el panel *JScrollPane* para que adopte la posición correcta. Para finalizar se actualiza el valor de la variable *línea*.

Como ya se ha comentado, este procedimiento se utilizará también cuando se complete una línea y se quiera pintar una nueva.

4.2.3- Método pintaNota()

Para todo el desarrollo gráfico de pintar las notas existe un método que se ocupa de esta tarea.

El método *pintaNota()* está considerado como la espina dorsal del programa ya que es el encargado directo de pintar notas, registrarlas y volverlas a pintar cuando se abre un archivo propio.

Recibirá datos de todos varios eventos: en este caso del evento de las figuras (una variable a true o a false), del evento que escucha al panel (un píxel) y en caso de ser necesario las modificaciones tonales.

Para dejarlo más claro se va a mostrar un diagrama con el orden que seguiría la aplicación para pintar una nota sin modificación tonal.

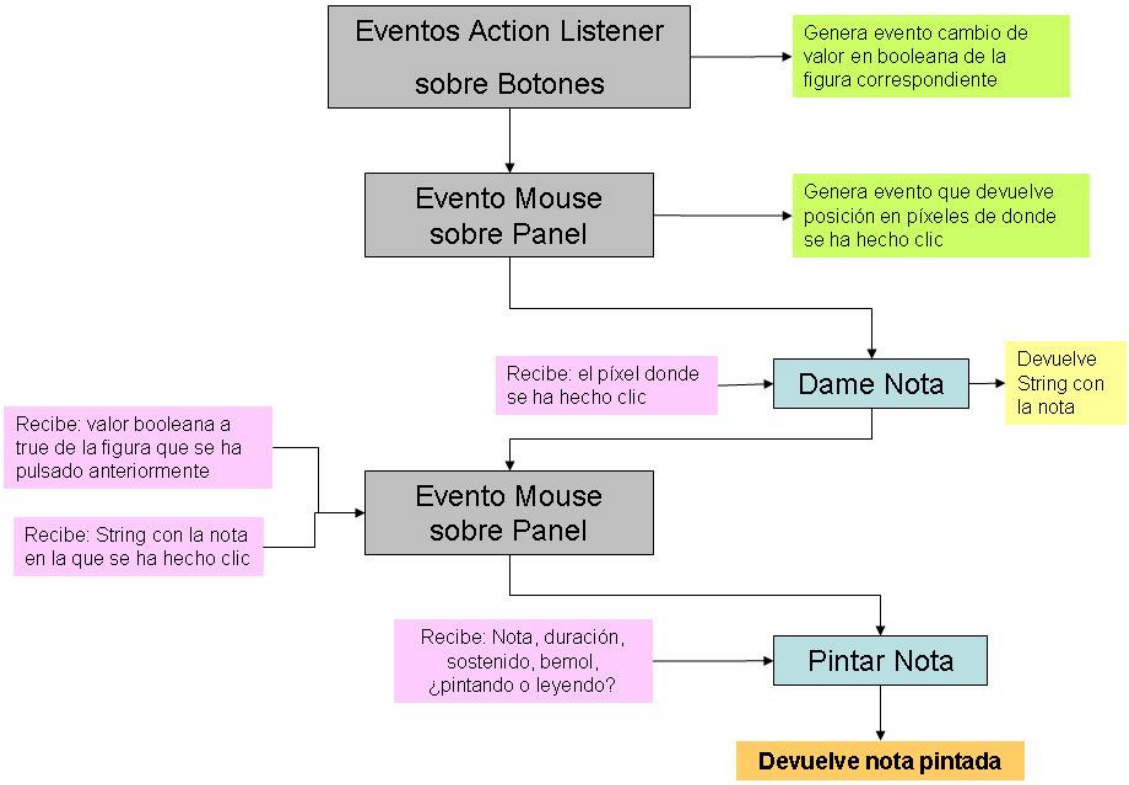

Figura 47: Orden de sucesos hasta llegar a pinta nota

Los eventos están en gris y para pintar una nota se necesitará de dos eventos distintos: Los botones con las figuras y hacer clic sobre el panel de edición en una parte concreta del pentagrama.

Lo que genera cada evento está en color verde. Como se ha dicho anteriormente, el evento del botón cambiará a *true* una variable *booleana* asociada a dicho botón. En el caso del evento del Mouse, devolverá la posición en píxeles sobre la que se ha hecho clic.

Previamente, cuando se genera el oyente asociado al panel de edición y a cualquiera de los botones de las figuras, se llamará al método *dameNota()* que recibirá la posición donde se ha hecho clic y devolverá un *String* con la nota a la que corresponde.

Para ello realizará un clasificador del tipo *if-else* y en función de los píxeles de los que consta el panel de edición y de manera relativa a la línea en la que se está escribiendo, se conseguirá el resultado.

```
     public String dameNota (){

       if (195>=yPos&&yPos>190)//Si la posición en la que se hace clic
está entre estos valores
          nota="do";//Devolverá un "do"
       else if (190>=yPos&&yPos>184)//Idem
          nota="re";
       else if (184>=yPos&&yPos>180)
          nota="mi";
       else if (180>=yPos&&yPos>174)
          nota="fa";
       else if (174>=yPos&&yPos>168)
          nota="sol";
       else if (168>=yPos&&yPos>162)
          nota="la";
       else if (162>=yPos&&yPos>155)
          nota="si";
       else if (155>=yPos&&yPos>149)
          nota="do'";
       else if (149>=yPos&&yPos>143)
          nota="re'";

       return nota;
    }
```

Figura 48: Método dameNota()

Posteriormente, se llamará al método *pintaNota()* dándole como atributos el *String* que devuelve *dameNota(),* la duración de la figura en la que se ha hecho clic y la operación que se realiza (escritura o lectura).

Es importante distinguir entre las operaciones que se realizan ya que si se está leyendo de un archivo, se tomarán algunas variables como línea y posición del archivo. A cambio si se está escribiendo, dichas variables se tendrán que ir actualizando por la aplicación. El último caso será el de borrado de figuras, en éste no se debe registrar ningún dato ya que estará aún en memoria.

A partir de aquí el método empezará a funcionar por si mismo. Es un método complicado que se apoya asimismo en otros cinco métodos más. El orden que sigue este método es el siguiente:

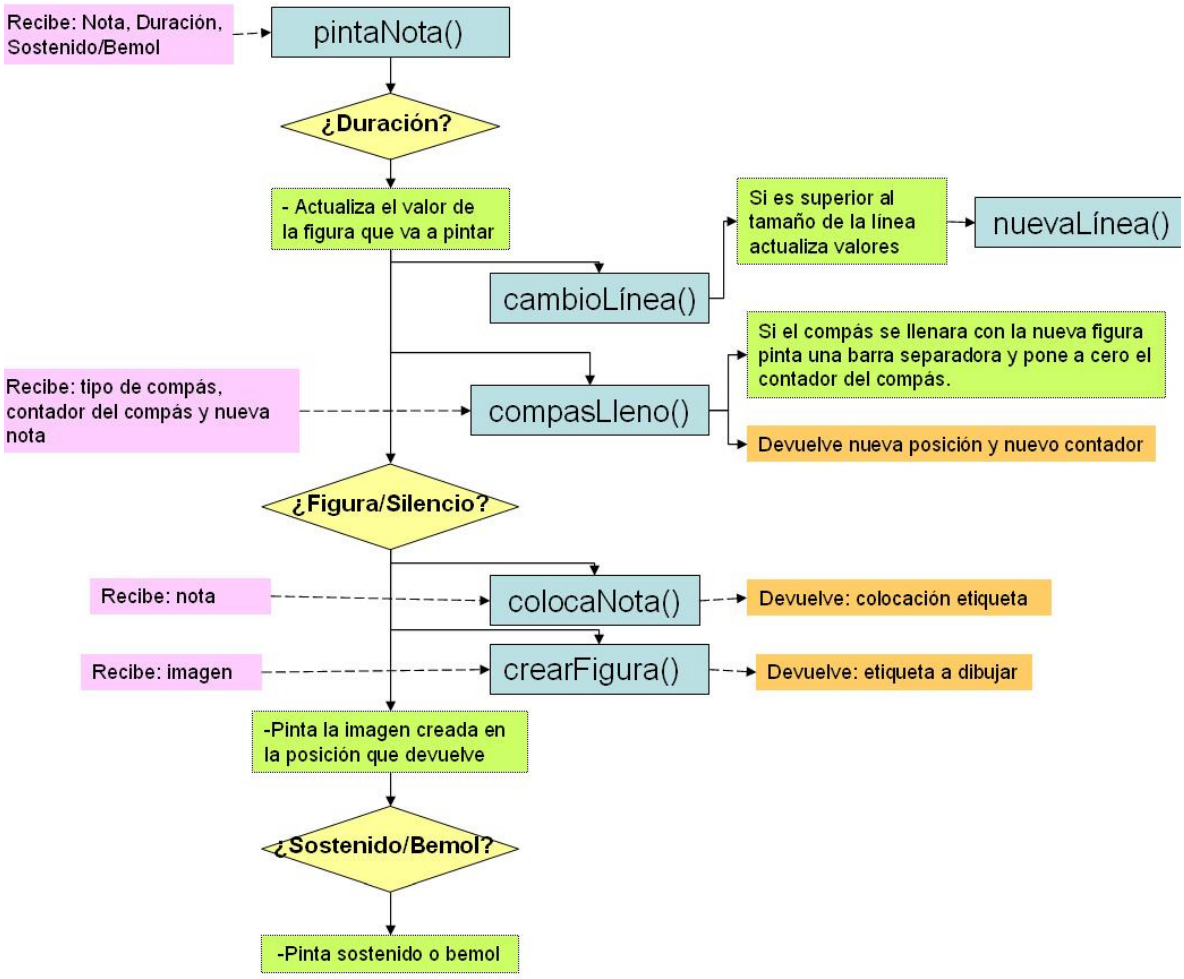

Figura 49: Método pintaNota()

El método *pintaNota(),* tras recibir estos atributos, realizará el proceso en función de la duración de la figura o silencio que reciba. Por ejemplo, si recibe una redonda, tomará una variable *int* de valor ocho y, en función de este valor comprobará si se ha llenado la línea (el método *cambioLinea()* comprobará si la posición en la que se pinta más el ancho de la figura es superior al tamaño del pentagrama, si es así, llamará al método *nuevaLinea()*).

El método *cambioLinea()* realiza una simple comprobación:

```
     public boolean cambioLinea(int posicion){

       if (posicion>880){//Si la posición supera el tamaño del
pentagrama
          nuevaLinea (TipoCompas);//Se crea una nueva línea
          contCompas=0;//Se reinicia el valor de cada compás
          return true;//Y se devuelve true al método pintaNota()
       }
       else{
          return false;
       }
    }
```

Figura 50: Método cambiaLinea()

Posteriormente comprobará si al pintar esta nueva figura, se llena un compás. Esto lo realizará el método *lleno()* al que se le pasará un contador con las notas que hay en el compás, la nueva figura y el tipo de compás. ¡!

```
   public int [] lleno (int contCompas, int posicion, int duracion,
boolean cambio){
     if (TipoCompas==24){//Si el compás es un dos por cuatro
        if(contCompas==4){//Y se han rellenado las cuatro partes
           if (!cambio){//No es un cambio de línea
              JLabel eSeparador=crearFigura (new ImageIcon
("Separador.gif"));//Dibujar línea separadora
              eSeparador.setBounds(posicion, 90+(linea-
1)*ipentagrama.getIconHeight(),
iSeparador.getIconWidth(),iSeparador.getIconHeight());//Emplazarla
              mPentagramas.add(eSeparador, JLayeredPane.DRAG_LAYER);
              posicion=posicion+iSeparador.getIconWidth();
//actualizar posición
              String barra="barra";
              registro.addElement(new Nota (barra, linea, posicion,
posicion+iSeparador.getIconHeight(), 0, false, false));
              contCompas=0; } } }

     if(TipoCompas==34){
        if (contCompas==6){ /*Se realizará el mismo proceso*/ } }
     else if (TipoCompas==44){
        if (contCompas==8){ /*Se realizará el mismo proceso*/  } }
     resultado[0]=contCompas;
     resultado[1]=posicion;
     return resultado;}//Devuelvo un array con la nueva posición y
contador
```

Figura 51: Método lleno()

En caso de que esté lleno, reiniciará el compás, pintará una línea de separación de compases y devolverá una nueva posición desde la que se debe continuar pintando.

Tras haberse explicado dos de los métodos en los que se apoya *pintaNota(),* se muestra el código correspondiente a la primera mitad de *pintaNota()* para las figuras redondas:

```
    public void pintaNota (String nota, int duracion, int ancho1,
boolean sostenido, boolean bemol, boolean pintar, boolean borrar){

      int Ydib=0;//Posición de la nota

      if(duracion==8){// Si es una redonda
         if (!pintar){//Si se está repintanto
            ancho1=posicion;}

         boolean
cambio=cambioLinea(posicion+ifRedonda.getIconWidth());//Se comprueba
si cambia de linea
         if (cambio) posicion=80;//Si cambia se pone la posicion a 0
         resultado=lleno(contCompas,
posicion+ifRedonda.getIconWidth(), duracion, cambio);//Se comprueba si
el compás está lleno

         contCompas=resultado[0]+8;//Se actualiza el contador del
compas
         posicion=resultado[1];
         Ydib=colocaNota(nota,8);//Se ve en qué píxel se dibuja
...
```

Figura 52: Código del método pintaNota(), parte 1

Llegados a este punto, se diferenciará entre silencio y figura ya que tienen una imagen distinta en cada caso. En función de este parámetro gracias al método *crearFigura()* del que se habló en la creación de la interfaz, se recibirá una etiqueta con la figura o el silencio que corresponde a la duración y se colocará en el punto exacto correspondiente a su nota (dicha colocación la dará *colocaNota()*).

Cabe destacar que el método *colocaNota()* será el encargado de pintar una línea auxiliar si la nota es un Do central.

```
    public int colocaNota (String nota, int duracion){

      int media=0;
      if (nota.equals("do")){ //Para la nota do
         media=192+(linea-1)*ipentagrama.getIconHeight()-21;//Se
pintará la figura en media
         JLabel ebarraDo=  crearFigura(new ImageIcon ("barraDo.gif"));
//Se necesitará una barra auxiliar
//En función de la figura se pintará en un sitio u otro
         if (duracion==8)
            ebarraDo.setBounds(posicion+49,media,
ibarraDo.getIconWidth(), ibarraDo.getIconHeight());
         else if (duracion==4) //Idem
         else if (duracion==2) //Idem
         else if (duracion==1) //Idem
         mPentagramas.add(ebarraDo, JLayeredPane.POPUP_LAYER);}
      else if (nota.equals("re")) //Para re, etc.
         media=187+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("mi"))
         media=181+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("fa"))
         media=177+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("sol"))
         media=171+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("la"))
         media=165+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("si"))
         media=159+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("do'"))
         media=153+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("re'"))
         media=147+(linea-1)*ipentagrama.getIconHeight()-21;
      else if (nota.equals("silencio"))
         media=153+(linea-1)*ipentagrama.getIconHeight();

      return media;
   }
```

Figura 53: Método colocaNota()

Para finalizar, el método *pintaNota()* hará una comprobación de si la figura que se está dibujando lleva asociada alguna modificación tonal (sostenido o bemol) y en caso de que así fuera la dibujaría en el lugar adecuado (es decir, a la misma altura que el cuerpo de la figura y con una separación lateral de la misma).

Será en ese punto cuando se actualice el contador de la posición y, en caso de ser necesario, se registrarán los datos en el vector correspondiente. Estos datos se utilizarán posteriormente para la creación de distintos tipos de archivos como se verá en el apartado 4.3.

A continuación se muestra la parte final del método *pintaNota().* Cabe destacar que solo se ha mostrado en esta memoria el proceso correspondiente a las figuras redondas pero que el proceso para el resto de figuras será igual salvo la variación de la variable *int duracion* y la figura que se dibujará en cada caso.

```
  ...
  if (nota.equals("silencio")){// Si es silencio
     JLabel esRedonda=crearFigura(new ImageIcon
("sRedonda.gif"));//Creo la etiqueta
     esRedonda.setBounds(posicion, Ydib,
isRedonda.getIconWidth(),isRedonda.getIconHeight());//Y la pinto
     mPentagramas.add(esRedonda, JLayeredPane.MODAL_LAYER);
  }
  else{//Si es figura sigo el proceso análogo
     JLabel efRedonda=crearFigura(new ImageIcon ("fRedonda.gif"));
     efRedonda.setBounds(posicion, Ydib,
ifRedonda.getIconWidth(),ifRedonda.getIconHeight());
     mPentagramas.add(efRedonda, JLayeredPane.MODAL_LAYER);}
  if (sostenido==true){
     JLabel eSostenido=crearFigura(new ImageIcon ("sostenido.gif"));
     eSostenido.setBounds(posicion+25,Ydib+7,
iSostenido.getIconWidth(), iSostenido.getIconHeight());
     mPentagramas.add(eSostenido, JLayeredPane.POPUP_LAYER);
  }
  else if (bemol==true){
     JLabel eBemol=crearFigura(new ImageIcon ("bemol.gif"));
     eBemol.setBounds(posicion+25,Ydib+7, iBemol.getIconWidth(),
iBemol.getIconHeight());
     mPentagramas.add(eBemol, JLayeredPane.POPUP_LAYER);
  }
  if (!borrar) registro.addElement(new Nota(nota, linea, posicion,
posicion+ifRedonda.getIconWidth(), 8, sostenido, bemol));
  posicion=posicion+ifRedonda.getIconWidth();}//Incremento la posición
en la anchura de la figura (es proporcional)
```

Figura 54: Código del método pintaNota(), parte 2

4.2.4- Reiniciar la Aplicación y Borrar las Figuras

Para explicar el borrado de figuras se deberá hacer de dos formas distintas, la primera está relacionada con su borrado a nivel gráfico, la otra es cómo se eliminarán los datos de las figuras borradas. En este capítulo se tratará su borrado gráfico y en el siguiente se explicará a nivel de estructura de datos.

El método para eliminar las figuras es reiniciar el panel para posteriormente volverlo a pintar. Se realiza en una clase interna, *OyenteEventoBorrar* que, al igual que las clases internas escuchadoras de botones, tendrá la siguiente estructura:

```
class OyenteEventoBorrar implements ActionListener{

   public void actionPerformed(ActionEvent evento){

     JButton  boton=(JButton) evento.getSource();

     if(boton==bFigura) {

      //Código correspondiente a botón bFigura (borrar figura)

     }
     else if (boton==bPentagrama){

     //Código correspondiente a borrar pentagrama

     }
   }
}
```

Figura 55: Ejemplo de código de una clase escuchadora

Para reiniciar el panel de edición el programa pedirá una confirmación previa, en caso de autorizarlo, se reinicializarán todas las variables, se borrará el Panel de tipo *JLayeredPane* y se repintará el contenedor *JFrame*.

Este procedimiento será el utilizado al hacer clic en Archivo>Reiniciar. Cuando se desea borrar (bien sea la última nota o la última línea del pentagrama), tras reiniciar el panel de edición y borrar el dato o los datos que han sido eliminados, se recorrerá un bucle *for* para cada dato del vector, llamando al método *pintaNota()* para cada figura que antes estaba representada (salvo las que han sido eliminadas).

4.3.- Implementación de los Módulos de Estructura de Datos

"La música es un medio incomparablemente poderoso, cuyo lenguaje sutil expresa los mil momentos diferentes del alma"

Piotr Ilich Chaikovski

Para la generación de archivos se necesita contar con la información que se ha introducido por la interfaz. Como se dijo en el capítulo de diseño del sistema, toda la información se almacena desde la clase principal, *Interfaz*, justo después de representarse gráficamente.

En este capítulo se hablará del funcionamiento de la unidad que almacena los datos, cómo se accede a ella, cómo se escriben los ficheros y qué estructura tienen los tres tipos de archivos que se generan: archivos propios *.mp*, archivos MusicXML y archivos MIDI. Por último, se hablará de la reproducción de los archivos MIDI.

4.3.1.- Unidad de Almacenamiento

Cada nota que se dibuja sobre el pentagrama tiene una serie de atributos.

Dichos atributos son: nota, duración, línea en la que es pintada, píxel inicial y píxel final en el que se dibuja, sostenido y bemol. Cada nota será almacenada en un objeto de la clase *Nota*.

Esta clase será capaz de abstraer las funciones y los atributos de cada nota; es decir, será capaz de definir *Nota* en términos de qué puede hacer y qué características lo distinguen de otros objetos.

De esta forma, la clase *Nota* tendrá los siete atributos que se han citado previamente (nota, duración, línea, ancho inicial, ancho final, sostenido, bemol). El primer atributo, nota, será de tipo *String*[4]*;* los cuatro siguientes atributos, duración, línea, ancho inicial y ancho final, serán del tipo *Integer*; para finalizar los dos últimos atributos, sostenido y bemol, serán variables de tipo *booleana*.

Cuando hay que referirse a las funciones de dicha clase se debe hablar de un único método que devuelve los siete atributos como una única cadena de *Strings* (los atributos están separados por comas). Será el método *toString()*.

En Java, la información suele ser guardada en *arrays*, esto suele ser suficiente para guardar tipos básicos de datos y objetos de determinadas clases cuyo número se sabe previamente. En este caso, como el número de notas no es sabida de antemano, se deberá utilizar un vector. La clase *Vector* es similar a un *array* solo que crece automáticamente conforme se le va introduciendo datos. Además, proporciona métodos adicionales para añadir, eliminar e insertar elementos en el índice que sea necesario.

[4] Una variable es un nombre que se asocia con una porción de la memoria del ordenador, en la que se guarda el valor asignado a dicha variable. Hay varios tipos de variables, las utilizadas aquí son: int (entero con signo), string (un texto cualquiera) y boolean (tiene dos valores true o false).

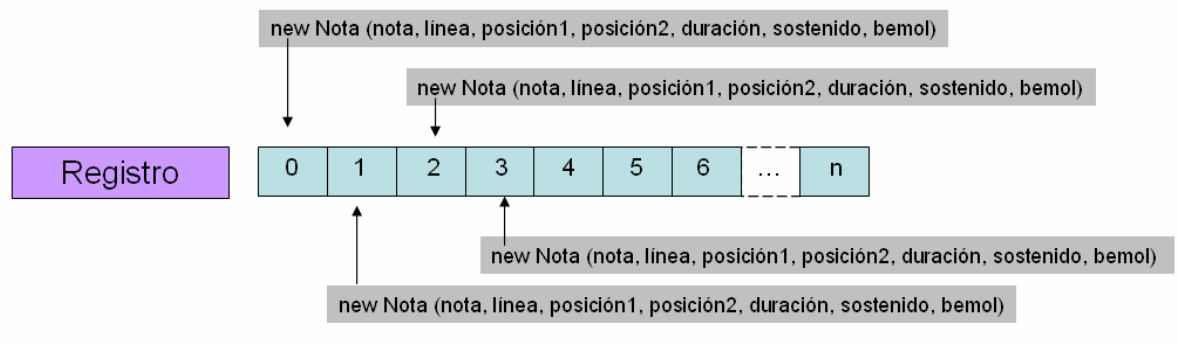

Figura 56: Almacenamiento en el vector de los distintos objetos de la clase Nota

Para su creación se necesita importar el paquete *java.util.**. Será creado en la clase principal, *Interfaz,* con tamaño inicial de dos objetos y con el nombre de *registro*. Gracias al método *addElement()*, después de pintar cada figura en el pentagrama, se añadirá al registro todos sus atributos creando una nueva instancia de la clase *Nota*.

Tan importante es tener los datos como poder rectificarlos y tener acceso a ellos. Por ejemplo, antes se ha descrito qué pasa cuando se borran las figuras, pero a nivel de gestión de datos se debe eliminar estos objetos del vector.

La forma de realizarla es sencilla, en el caso de eliminar la última nota, se utilizará el método, que proporciona la clase *Vector, removeElementAt()* y se indica en el paréntesis el último elemento del vector.

A cambio, cuando se desea eliminar la última línea del pentagrama se deberá recorrer todo el vector (con un bucle *for*) y para todos los elementos que estén en la última línea, aplicarles el método anteriormente mencionado: *removeElementAt().*

Estas dos últimas acciones son desarrolladas en la clase interna *OyenteEventoBorrar* que implementa un *ActionListener* y realiza una u otra opción en función de si el evento se escucha en un botón o en el otro.

Por último, ya que el vector se crea y se modifica en la clase principal, *Interfaz*, para tener acceso al vector desde otras clases, se ha declarado el método *getMemory()* que devuelve el vector *registro*.

4.3.2.- Creación y Lectura de Archivos

En Java, la salida de datos se realiza mediante un flujo de salida. Para realizar la creación de archivos, se abre un flujo de salida y se va escribiendo en él toda la información que se desee generar de manera sucesiva.

Análogamente, cuando se requieren datos del exterior, se abre un flujo de entrada y se va leyendo sucesivamente la información del fichero hasta el final.

Un flujo en Java representa un objeto que sirve para hacer entradas y salidas de datos, un canal de información que puede trabajar con bytes o con caracteres. En este caso se utilizarán flujos con caracteres del tipo *BufferedReader* y *BufferedWriter*. Como se busca tratar archivos se utilizará *File* como representación de dicho archivo.

En el caso de la escritura de archivos se abrirá inicialmente *fileDialog* para seleccionar dónde guardar el archivo y su nombre. Posteriormente almacenará línea a línea lo que se desee en función del tipo de archivo que se deseé generar, gracias al método *write().* Después de cada línea que se quiera guardar se llamará al método *newLine().*

Por último se deberá cerrar el flujo de datos y capturar las excepciones correspondientes con el método *try-catch* (se capturará la excepción *IOException*).

En cambio, para el caso de la lectura de datos, se creará otro *fileDialog* (esta vez del tipo *LOAD* en lugar de *SAVE*) que dirá la ruta y el nombre del archivo.

El flujo de datos *BufferedReader* creará un nuevo *FileReader* con los datos que devuelve el *fileDialog* y, tras comprobar que hay datos, irá leyendo línea a línea todo el archivo gracias al método *readLine().* Este método devolverá un *String.*

También se capturarán con el método *try-catch* la excepción *IOException.*

4.3.3.- Archivos en Formato Propio (.mp)

Para explicar la distinta forma y estructura que tiene cada tipo de archivos se utilizará un ejemplo básico. Será una escala en la que se introducirán silencios y figuras de distinta duración dentro de un compás de tres por cuatro.

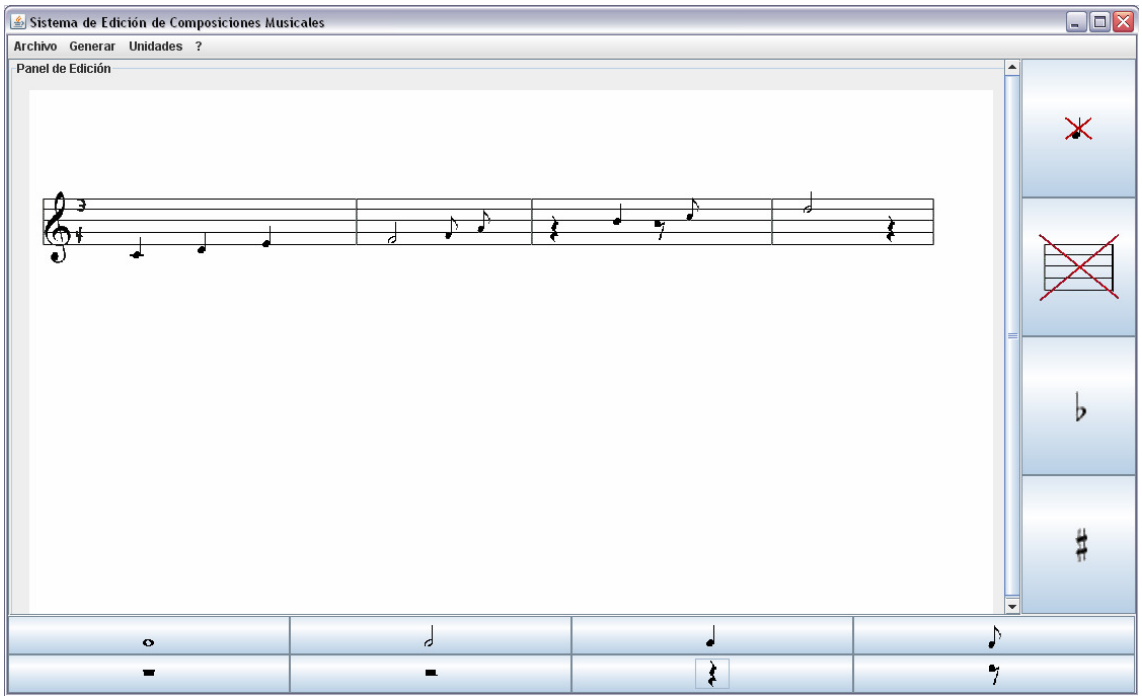

Figura 57: Ejemplo “Escala”

La aplicación utilizará el ya nombrado *BufferedWriter* para escribir la información con el orden que se ha considerado adecuado para su posterior lectura. En la primera línea se registrará el compás en el que se ha escrito la melodía, en este caso un tres por cuatro. En cada una de las siguientes líneas se almacenan como una cadena de *Strings*, todos los atributos de cada nota con este orden: nota (si es silencio pondrá silencio), duración (8 redonda, 4 blanca, 2 negra, 1 corchea), sostenido, bemol, línea, ancho1 y ancho2.

Cabe destacar que si se genera una barra de separación entre compases, ésta, se guardará como si fuera una nota con duración cero. A nivel de creación de otros archivos no influye ya que como se desconoce, se ignora. Sólo es útil para cargar el archivo más adelante.

El código correspondiente se encuentra en la clase *OyenteEventoGuardar* que, a parte de generar los archivos propios *.mp,* también creará los *MusicXML* que se explicarán a continuación. Dicha clase, por tanto, escuchará dos eventos distintos. Algunas partes del código serán comunes:

```
    public class OyenteEventoGuardar implements ActionListener{

      private Interfaz i=new Interfaz();
      protected FileDialog fdgrabar=null;
       public OyenteEventoGuardar(Interfaz i) {
         this.i=i;}

       public void actionPerformed(ActionEvent evento){

         File Datos;//Se crea el fichero
         BufferedWriter bw;//Instancia del objeto bw de tipo
BufferedWriter
         Vector registro=i.getMemory();
         JFrame marco=new JFrame();
         String Pentagrama=null;
         JMenuItem opcion=(JMenuItem) evento.getSource();
         fdgrabar=new FileDialog(marco, "Guardar Fichero",
FileDialog.SAVE);   //dialogo para grabar la grabacion
         fdgrabar.setVisible(true);//Se muestra la ventana grabación
         Pentagrama=new String(fdgrabar.getDirectory() +
fdgrabar.getFile());//Se pasa a un String la dirección donde se graba
el archivo
```

Figura 58: Clase OyenteEventoGuardar, parte común

A partir de este punto comienza la tarea de escribir el archivo propio. La forma en la que se realiza es la siguiente:

```
   if(opcion==i.guardar) {
      try{
         if(!Pentagrama)
            Datos=new File("Pentagrama.mp");// Creo el fichero
         else
            Datos=new File(Pentagrama+".mp");
         bw=new BufferedWriter(new FileWriter(Datos));
         if (Datos.exists()){//Si el fichero ha sido creado
            bw.write(""+i.getCompas());
            bw.newLine();
            for(int j=0;j<registro.size();j++){
               bw.write(""+registro.elementAt(j).toString());
               bw.newLine();
               System.out.println("Registro=
"+registro.elementAt(j).toString());
            } }
         bw.close();}//try
          catch (IOException e) {//Capturo excepción
            e.printStackTrace();}// catch
   }//if
```

Figura 59: Clase OyenteEventoGuardar, creación de archivos .mp

Como se ha dicho con anterioridad, se generará un archivo *.mp* que tendrá en la primera línea el compas y posteriormente las notas en distintas líneas. Para el ejemplo anterior el archivo resultante será el siguiente:

```
34
do,2,false,false,1,112,144
re,2,false,false,1,176,208
mi,2,false,false,1,240,272
barra,0,false,false,1,349,465
fa,4,false,false,1,349,412
sol,1,false,false,1,428,444
la,1,false,false,1,460,476
barra,0,false,false,1,522,638
silencio,2,false,false,1,522,554
si,2,false,false,1,586,618
silencio,1,false,false,1,634,650
do',1,false,false,1,666,682
barra,0,false,false,1,759,875
re',4,false,false,1,759,822
silencio,2,false,false,1,854,886
```

Figura 60: Archivo 'Escala.mp'

De esta forma, al leer los archivos propios se seguirá el proceso escrito en pseudocódigo de la figura siguiente:

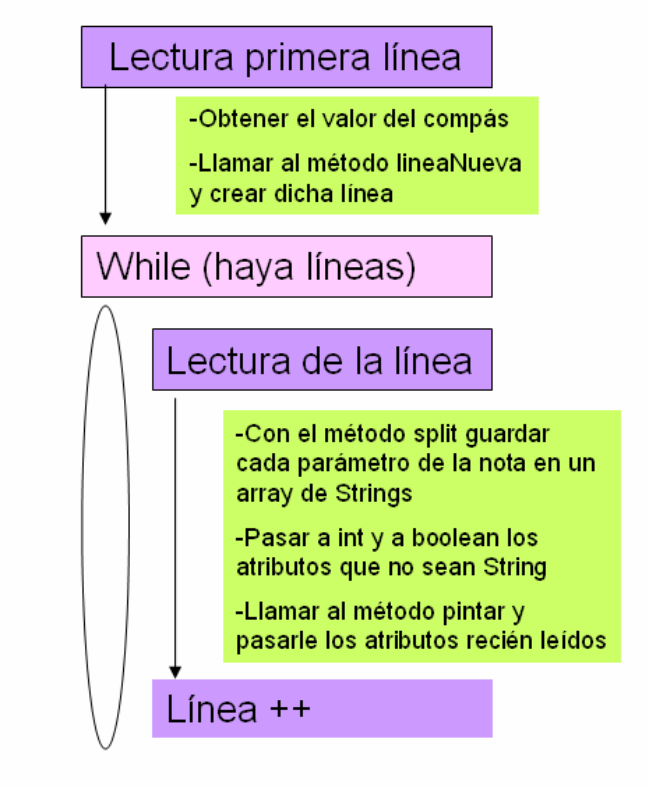

Figura 61: Funcionamiento de la Lectura de archivos propios

En resumen, la primera línea da información del tipo de compás y permite pintar el pentagrama. Las demás líneas se tratarán una a una y para cada línea se llamará al método *pintarNota()* con los atributos necesarios.

Para tener los atributos se leerá cada nota, se guardará cada atributo en una posición de un array gracias al método *split* y se pasará al formato adecuado (si es *Integer* con el método *parseInt()*, y si es *boolean* con el método *parseBoolean()*).

Se finalizará la clase cerrando el *buffer* que lee datos.

4.3.4.- Archivos MusicXML

Como se comentó en el Capítulo 2 del Estado del Arte, dentro de los protocolos de comunicación de datos, actualmente MusicXML es uno de los más nombrados dada la cantidad de matices que permite recopilar en el almacenamiento de datos musicales.

La forma de almacenar los datos en este tipo de archivos es similar a los archivos propios (con un *BufferedWriter*) pero siguiendo una estructura más compleja.

Inicialmente se almacenarán los datos relativos a la cabecera de los archivos MusicXML que hace referencia a datos tan importantes como la versión (actualmente 1.0), codificación, si necesita o no un archivo dtd y en caso de ser necesario dónde se puede encontrar dicho dtd de carácter público.

```
  if(opcion == i.xml) {

   try{
      if(Pentagrama==null)
         Datos = new File("Pentagrama.mxl" );// Se crea el fichero
      else
         Datos = new File(Pentagrama+".mxl" );
      bw = new BufferedWriter(new FileWriter(Datos));
      if (Datos.exists()){//Si el fichero ha sido creado
         bw.write("<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"no\"?>");//se escribe
         bw.newLine();
         bw.write("<!DOCTYPE score-partwise PUBLIC\"-//Recordare//DTD
MusicXML 2.0
Partwise//EN\"\"http://www.musicxml.org/dtds/partwise.dtd\">");
         bw.newLine();
         bw.write("<score-partwise version=\"2.0\">");
         bw.newLine();
         bw.write("<movement-title>"+Pentagrama+"</movement-title>");
```

Figura 62: Clase OyenteEventoGuardar, creación de archivos MusicXML_1

Luego se declararán los datos relativos a la melodía a almacenar tales como pentagrama, parte, identificación de todos esos parámetros, nombre del archivo, etc.

De ahí se pasará a almacenar el compás entre etiquetas *<beats>* y *<beat-type>,* la clave (que se almacena como *<clef>,* y dentro, *<sign>* para la nota y *<line>* para la línea en la que se dibuja), las notas y sus atributos.

```
  int compas=i.getCompas();
  if (compas==44){    //Compás
     bw.write("<beats>4</beats>");
     bw.newLine();
     bw.write("<beat-type>4</beat-type>");
     bw.newLine();
  }
  else if (compas==34){
     bw.write("<beats>3</beats>");
     bw.newLine();
     bw.write("<beat-type>4</beat-type>");
     bw.newLine();
  }
  else if (compas==24){
     bw.write("<beats>2</beats>");
     bw.newLine();
     bw.write("<beat-type>4</beat-type>");
     bw.newLine();
  }
bw.write("</time>");
bw.newLine();
bw.write("<clef>");//Clave
bw.newLine();
bw.write("<sign>G</sign>");//Sol
bw.newLine();
bw.write("<line>2</line>");//En la segunda linea
bw.newLine();
bw.write("</clef>");
bw.newLine();
bw.write("</attributes>");
```

Figura 63: Clase OyenteEventoGuardar, creación de archivos MusicXML_2

Para cada nota se almacena la frecuencia (con el paso, la octava, y la alteración en caso de que sea sostenido o bemol), la duración y el tipo (*whole*).

Para rellenarlo, se declaran lo atributos del pentagrama de manera genérica y, tras eso, se recorre un bucle for para el vector de notas, y en función de la nota, duración y alteración almacenada, se rellena de una forma u otra.

```
for(int j=0;j<registro.size();j++){
//Se leen datos:
   String cadena= registro.elementAt(j).toString();
   String[] melodia = cadena.split(",");
   int duracion=Integer.parseInt(melodia[1]);
   boolean sostenido= Boolean.parseBoolean(melodia[2]);
   boolean bemol= Boolean.parseBoolean(melodia[3]);
   bw.write("<note>");
   bw.newLine();
   if (melodia[0].equals("do")){ //Poner aquí nota
      bw.write("<pitch>");
      bw.newLine();
      bw.write("<step>C</step>");
      bw.newLine();
      bw.write("<octave>4</octave>");//Octava C central
      bw.newLine();
   }
 //Igual para el resto de notas
   else if (melodia[0].equals("silencio")){
      bw.write("<rest/>");
      bw.newLine();
   }
   if(bemol){
      bw.write("<alter>-1</alter>");//para bemol
      bw.newLine();
   }
   else if (sostenido){
      bw.write("<alter>1</alter>");//para sostenido
      bw.newLine();
   }
   if (!melodia[0].equals("silencio")){
      bw.write("</pitch>");
      bw.newLine();
   }
   if (duracion==1){//Poner duración, esta es redonda
      bw.write("<duration>1</duration>");
      bw.newLine();
   }//Igual para blanca (2), negra (4), corchea (8)
```

Figura 64: Clase OyenteEventoGuardar, creación de archivos MusicXML_3

A continuación se muestra el documento creado para el ejemplo anterior.

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<!DOCTYPE score-partwise PUBLIC"-
//Recordare//DTD MusicXML 2.0
Partwise//EN""http://www.musicxml.org/dtd
s/partwise.dtd">
<score-partwise version="2.0">
<movement-title>Escala</movement-title>
<part-list>
<score-part id="P1">
<part-name>Part 1</part-name>
</score-part>
</part-list>
<part id="P1">
<measure number="1">
<attributes>
<divisions>1</divisions>
<key>
<fifths>0</fifths>
</key>
<clef>
<sign>G</sign>
<line>2</line>
</clef>
</attributes>
<note>
<pitch>
<step>C</step>
<octave>4</octave>
</pitch>
<duration>2</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>D</step>
<octave>4</octave>
</pitch>
<duration>2</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>E</step>
<octave>4</octave>
</pitch>
<duration>2</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>F</step>
<octave>4</octave>
</pitch>
<duration>4</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>G</step>
<octave>4</octave>
</pitch>
<duration>1</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>A</step>
<octave>4</octave>
</pitch>
<duration>1</duration>
<type>whole</type>
</note>
<note>
<rest/>
<duration>2</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>B</step>
<octave>4</octave>
</pitch>
<duration>2</duration>
<type>whole</type>
</note>
<note>
<rest/>
<duration>1</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>C</step>
<octave>5</octave>
</pitch>
<duration>1</duration>
<type>whole</type>
</note>
<note>
<pitch>
<step>D</step>
<octave>5</octave>
</pitch>
<duration>4</duration>
<type>whole</type>
</note>
<note>
<rest/>
<duration>2</duration>
<type>whole</type>
</note>
</measure>
</part>
</score-partwise>
```

Figura 65: Archivo MusicXML ‘Escala.mxl’

4.3.5.- Archivos MIDI

Este es el último tipo de archivos que genera el Sistema de Edición de Composiciones Musicales desarrollado. Para este caso no se utilizará ningún flujo de salida de tipo *BufferedWriter* sino que habrá que apoyarse en la librería externa JMusic para obtener el archivo MIDI correctamente implementado.

Se generaran los archivos al escuchar cualquiera de los dieciocho botones que realizan esta acción. Cada botón dará las opciones elegidas por el usuario para generar el archivo, esto es, instrumento y tempo, entre las distintas opciones: violín, guitarra, clarinete, flauta, piano, xilófono; y tempo rápido, medio o lento para cada caso. De esta forma, la clase OyenteEventoGenerar escuchará dieciocho eventos distintos.

No obstante la forma de generar el archivo MIDI será igual, solo variará al crear la parte y al asignar un tempo al pentagrama. El procedimiento será el mostrado en la figura siguiente:

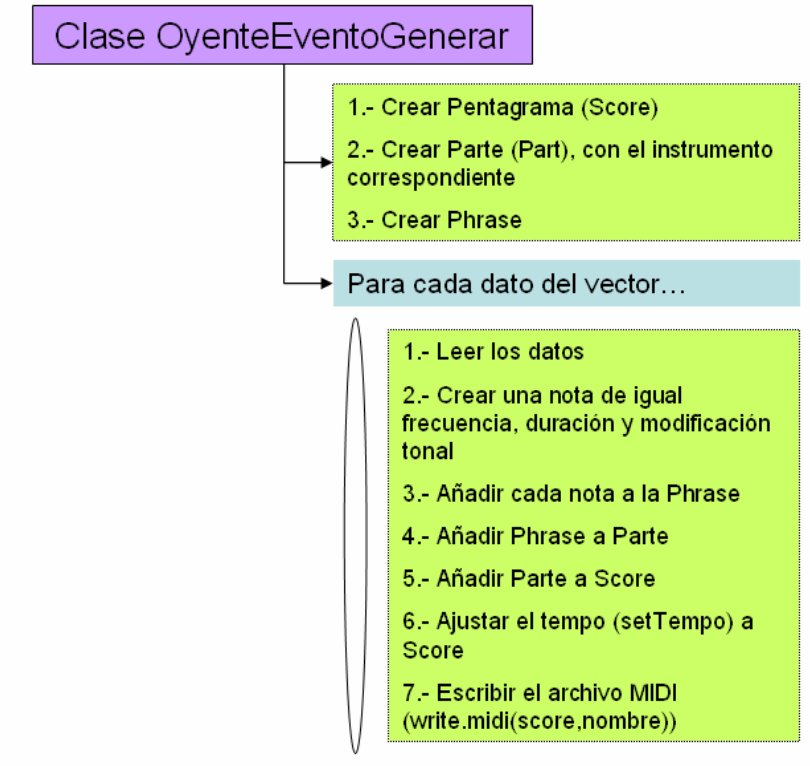

Figura 66: Escritura de un Archivo MIDI

De esta forma se pueden reconocer dos pasos principales en la creación de una melodía: cuando se crean los parámetros por orden jerárquico (pentagrama, parte, frase, nota) y cuando se añaden unos a otros. Para finalizar se configura el tempo y se escribe el archivo MIDI gracias al método *write.midi()*.

```
    class OyenteEventoGenerar implements ActionListener, JMC{

      public Interfaz i = new Interfaz();
      String nota=null;
      int duracion;
      protected FileDialog fdgrabar =null;
      String Pentagrama=null;
      JFrame marco=new JFrame();

       public OyenteEventoGenerar(Interfaz i) {
         this.i = i;}

       public void actionPerformed(ActionEvent evento){
         JMenuItem  opcion = (JMenuItem) evento.getSource();
         Score s = new Score("Partiturilla");
         Pentagrama = "Pentagrama";

         if(opcion == i.lflauta) {
            try{
               Vector registro = i.getMemory();
               Part p = new Part("Flute", FLUTE, 0);
               Phrase phr = new Phrase("Chromatic Scale", 0.0);

               Part pa=crear(registro, phr, p);
               s.addPart(pa);//empaqueta la parte en un pentagrama
               s.setTempo(100);
                  Write.midi(s, Pentagrama+".mid" );} //graba el
pentagrama en un archivo MIDI
                catch (Exception e) {//Capturar excepción
                  e.printStackTrace(); }}// catch

// Se sigue el mismo proceso para los restantes 17 eventos
```

Figura 67: Código OyenteEventoGenerar, parte 1

Como se puede observar, para la creación de las partes se utiliza el método *crear()* (sombreado en negrita en la figura anterior). Este método configura cada nota con la duración, frecuencia y modificación tonal adecuada. Cabe destacar que la forma de escribir cada nota es similar a la utilizada al escribir las notas en los archivos MusicXML.

Por ello se identifica el primer atributo del vector de objetos *Nota* (es decir, un *String* con la nota) y el segundo atributo (duración). A partir de ahí, se seguirá una estructura de control del tipo *if-else* con las distintas opciones (ya sea de notas, silencios, modificaciones tonales y duraciones) dando para cada caso una nota distinta.

Por ejemplo, si se leerían todos los datos de la siguiente forma:

```
public Part crear(Vector registro, Phrase phr, Part p){
  for(int j=0;j<registro.size();j++){

     String[] suena = registro.elementAt(j).toString().split(",");
     boolean sostenido= Boolean.parseBoolean(suena[2]);
     boolean bemol= Boolean.parseBoolean(suena[3]);
```

Figura 68: Código OyenteEventoGenerar, parte 2

Si se quisiera pintar un silencio, se haría lo que se muestra a continuación:

```
 if(sostenido==false&bemol==false){
    if (suena[0].equals("silencio")){
       if(suena[1].equals("8")){
          Note n = new Note(REST,SEMIBREVE);
          phr.addNote(n);//Pone la nota dentro de la frase
       }
       else if (suena[1].equals("4")){
          Note n = new Note(REST,MINIM);
          phr.addNote(n);
       }
       else if (suena[1].equals("2")){
          Note n = new Note(REST,CROTCHET);
          phr.addNote(n);
       }
       else if (suena[1].equals("1")){
          Note n = new Note(REST,QUAVER);
          phr.addNote(n);
       }
    }
```

Figura 69: Código OyenteEventoGenerar, parte 3

A cambio, si se quisiera dibujar un Do sostenido, seguiría la siguiente estructura de control:

```
  else if(sostenido==true&bemol==false){
     if (suena[0].equals("do")){
        if(suena[1].equals("8")){
           Note n = new Note(CS4,SEMIBREVE);
           phr.addNote(n);//Pone la nota dentro de la frase
        }
        else if (suena[1].equals("4")){
           Note n = new Note(CS4,MINIM);
           phr.addNote(n);
        }
        else if (suena[1].equals("2")){
           Note n = new Note(CS4,CROTCHET);
           phr.addNote(n);
        }
        else if (suena[1].equals("1")){
           Note n = new Note(CS4,QUAVER);
           phr.addNote(n);
        }
     }
p.addPhrase(phr);//Introduce la frase en una parte
return p;
}
```

Figura 70: Código OyenteEventoGenerar, parte 4

Y por último, como se puede ver en la figura anterior, devolvería la parte correspondiente. El resultado, como cabe esperar, es un archivo MIDI que se podrá reproducir posteriormente.

4.3.6.- Reproductor de Archivos MIDI

Para la realización del reproductor de archivos MIDI, J2SE ofrece la tecnología suficiente para realizarlo sin tener que hacer uso de bibliotecas externas [2].

Un dispositivo software que reproduce una secuencia MIDI es conocido como un *sequencer*. Una secuencia MIDI contiene listas con marca de tiempo de datos MIDI, estas deben ser leídas desde un archivo MIDI estándar. La mayoría de *sequencers* también proporcionan funciones para crear y editar las secuencias. La interfaz *Sequencer* incluye otros métodos para las secuencias MIDI básicas.

Algunas de estas operaciones son:

- Obtener la secuencia de un archivo MIDI
- Empezar o detener la reproducción de dicho archivo
- Moverse hasta un punto determinado de la secuencia
- Modificar el tempo de la reproducción

La forma en la que es utilizada por el Sistema de Edición es en una clase externa (*OyenteEventoReproducir*) al escuchar el evento que genera el botón reproducir asociado a dicha clase. De esta forma se creará el *sequencer,* se abrirá; se creará la secuencia, se le pasará como parámetro el archivo MIDI creado anteriormente, y se añadirá la secuencia al secuenciador.

Por último se podrá reproducir. Es importante destacar que en este caso, como en casos anteriores, se capturarán con el método t*ry-catch* las distintas excepciones que se pueden generar: *MidiUnavailableException, InvalidMidiDataException* e *IOException.*

A continuación se muestra la parte más relevante de esta clase.

```
try{
   Sequencer sm_sequencer = MidiSystem.getSequencer();
   sm_sequencer.open();
   File midiFile = new File(titulo);
   Sequence sequence = MidiSystem.getSequence(midiFile);

   sm_sequencer.setSequence(sequence);
   sm_sequencer.start();
}
    catch (MidiUnavailableException e){
      e.printStackTrace();}
    catch (InvalidMidiDataException e){
      e.printStackTrace();}
    catch (IOException e){
      e.printStackTrace();}
```

Figura 71: Código OyenteEventoReproducir

5.- Conclusiones y Trabajos Futuros

5.1.- Conclusiones

"La música constituye una revelación más alta que ninguna filosofía"

Ludwig Van Beethoven

Antes de empezar un proyecto es importante crear y declarar una serie de objetivos que se deben cumplir. Dichos objetivos fueron explicitados en el capítulo de diseño del sistema, mientras que el apartado de implementación trató de dar cuenta de los pasos seguidos para su logro. Corresponde a este apartado someter a evaluación dichos objetivos para poder concluir su correcta consecución.

El objetivo principal del proyecto consistía en el diseño y desarrollo de un programa que constituyera una herramienta didáctica. Para ello, se asociaron una serie de contenidos musicales que se consideraron importantes tanto para el público seleccionado como para su nivel musical.

Como se ha expuesto a lo largo de la memoria, se han implementado las distintas duraciones de las figuras básicas (de la redonda a la corchea) en la barra de botones declarada en la parte inferior. También se han aplicado modificaciones tonales (en la barra de la derecha). Y a lo largo de la barra de menú se han situado los distintos compases cuaternarios (dos por cuatro, tres por cuatro, cuatro por cuarto), las distintas familias musicales (viento, cuerda, percusión, cuerda percutida) y tres tempos distintos.

Evidentemente, la adquisición de estos conceptos es también ampliable a estudiantes de otros niveles de música, si bien no era el objetivo inicial.

Para la consecución del objetivo principal, se fijo un segundo objetivo radicado en la obtención de una herramienta de fácil manejo, dado el carácter eminentemente didáctico de la aplicación y el público al que va dirigido. Fue este objetivo el que guió y determinó el diseño de la interfaz que posteriormente fue implementada, con el apoyo de las librerías básicas de Swing y AWT que ofrece Java.

En la siguiente imagen se puede ver la consecución de dicho objetivo. Como se pensó en el diseño inicial se encontrará en la parte superior de la interfaz gráfica una barra de menú, en la parte central el panel de edición donde se podrán dibujar, guardar y modificar los pentagramas. Por último se pueden observar las dos barras de herramientas (parte inferior y parte derecha) que servirán para pintar las distintas figuras y sus silencios asociados.

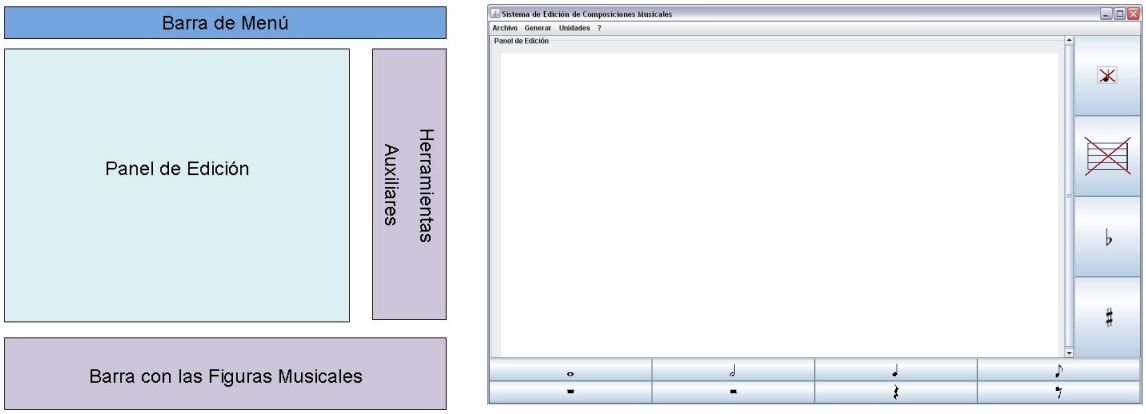

Figura 72: Comparación entre el diseño y la interfaz

Asimismo, la aplicación cumplirá los principios de interactividad gracias a la sencillez de utilización a la hora de pintar figuras o generar archivos MIDI y un diseño adecuado para el nivel de los usuarios haciendo accesible dicha aplicación.

Este sistema es una herramienta básica con un notado carácter didáctico que servirá de punto de partida para ampliaciones. Dichas ampliaciones serán explicadas en el apartado siguiente.

Se busca, en definitiva, que el usuario pueda desde el principio aprovecharse de las ventajas que ofrece la programación a la educación, la didáctica y la creación de música. Se confía en que este sistema no sea más que el comienzo.

5.2.- Trabajos Futuros

"La música comienza donde acaba el lenguaje"
Ernst Theodor Amadeus Hoffmann

Habitualmente se da por concluido un proyecto cuando éste obtiene por resultado la consecución de los objetivos que inicialmente se habían planteado. No obstante, el final de un proyecto también puede ser el punto de partida para otros nuevos: al fin y al cabo, el proyecto inicial siempre puede ser mejorado, seguir creciendo en la cabeza del creador y en la capacidad de crítica de todas las personas que lo conocen y utilizan.

Por ello, pese a haber alcanzado los objetivos fijados para el Sistema de Edición de Composiciones Musicales, conviene apuntar, a modo de cierre de la presente memoria, algunas de las posibles líneas de mejora que se vislumbran como trabajos futuros, en el anhelo de que nuestro proyecto nunca deje de crecer, de que este final sea también principio.

Muchas de las mejoras que se van a proponer harían que el programa fuera extensible aun público más amplio, de forma que pueda servir como herramienta educativa a estudiantes de hasta grado medio de cualquier instrumento.

Para ello, se intentarían incluir tanto en la interfaz gráfica como en la implementación lógica y generación de distintos archivos musicales, nuevos conceptos que son utilizados a partir de cierto nivel de conocimiento musical. Como no se desea perder la posibilidad de crear una interfaz fácilmente utilizable por estudiantes de grado elemental, el diseño de la interfaz sería prácticamente igual.

En este terreno, sería interesante aumentar los conceptos musicales de la barra de herramientas de la derecha e incluir conceptos como el puntillo y la ligadura.

Figura 73: Ejemplos del Puntillo y la Ligadura

Otra posibilidad que ampliaría el nivel de la herramienta didáctica sería contemplar la opción de dar a elegir al usuario entre distintas claves musicales, como la clave de Fa y la clave de Do, ambas bastante utilizadas en determinados instrumentos y cuyo conocimiento se exige en todo grado medio de conservatorio donde, en las clases de solfeo, se pasa a estudiar y solfear en dichas claves.

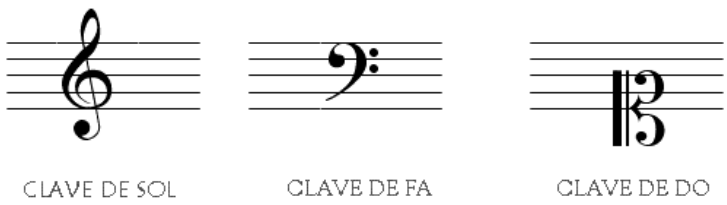

Figura 74: Claves musicales

Dentro de la barra de herramientas inferior podría incrementarse el número de figuras musicales. En el diseño inicial se plantearon las principales: blanca, redonda, negra y corchea. Pero para generar archivos MIDI se pueden usar las semicorcheas y en el caso de los archivos MusicXML se pueden hacer cuantas divisiones se quieran, incluyendo fusas.

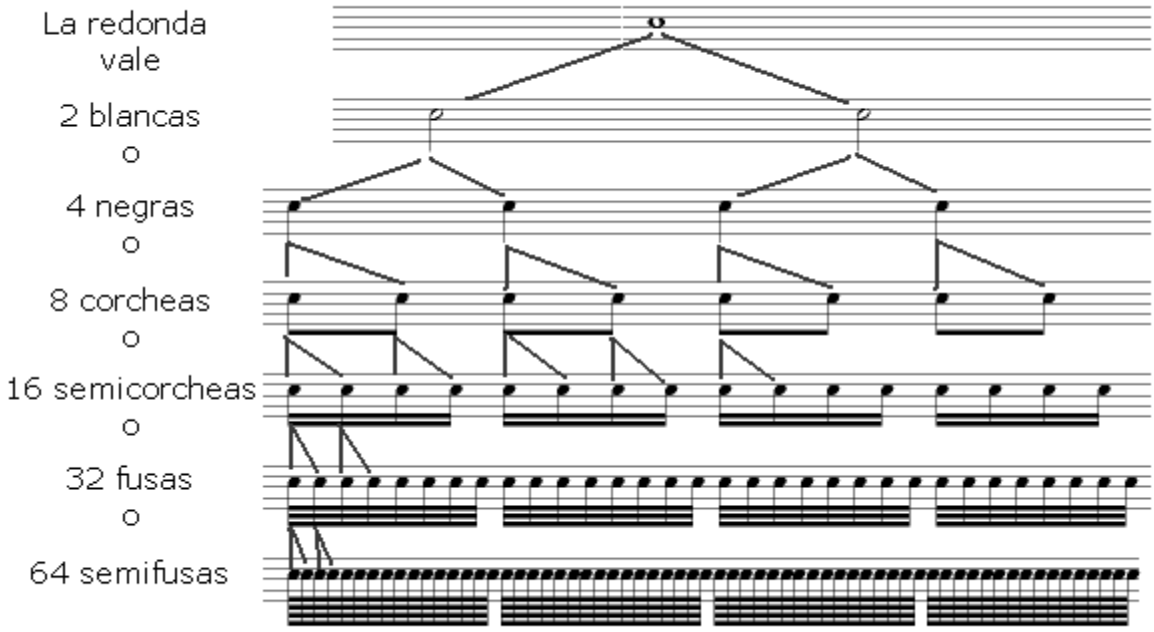

Figura 75: Valor de las Figuras

Dentro de la línea de ampliación de los contenidos y conceptos musicales de la herramienta didáctica, otra posible mejora sería ampliar la capacidad de elección en los tipos de compases (ya que en este proyecto solo se han implementado los cuaternarios).

Más allá de este campo, quizá, una de las mejoras más interesantes sería pasar de un editor de partituras monovoz a un editor para varias voces. Para poder mantener la relación de las notas entre las distintas voces, se mostrarían los pentagramas de las voces agrupadas en el panel de edición principal.

De esta forma, cada voz generaría un vector de datos y al generar archivos se juntarían todos los vectores para poder crear archivos multivoz.

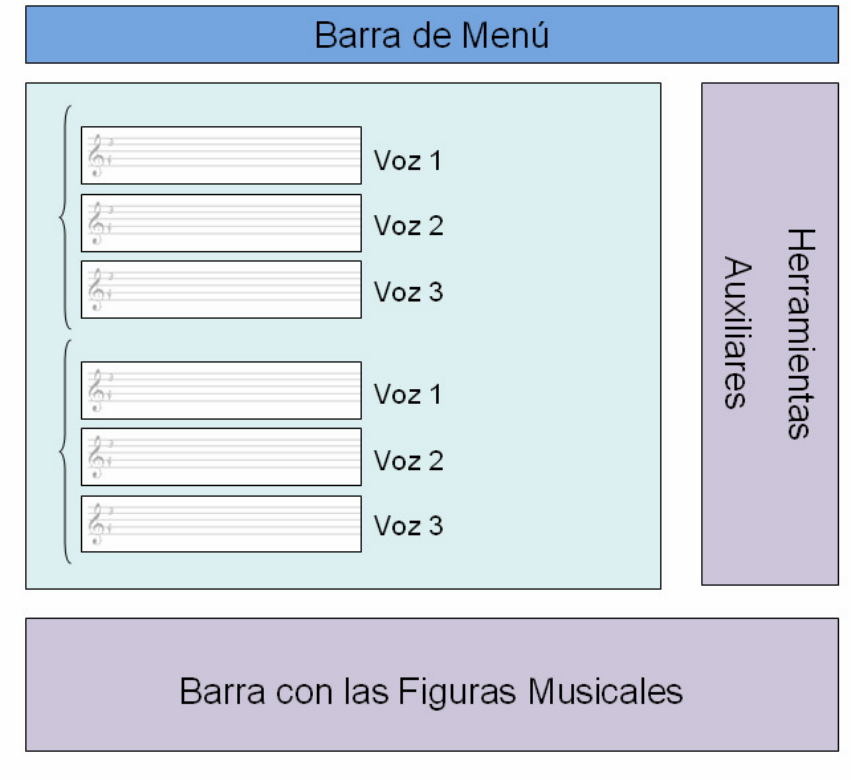

Figura 76: Posible diseño de una futura interfaz

Otro tipo de mejoras posibles, que no se encuentran relacionadas con el diseño, son aquellas relacionadas con las unidades externas que se pueden incluir a la aplicación. Dichas unidades se encuentran actualmente en desarrollo en otros Proyectos Fin de Carrera.

La implementación presente del programa ha contado con su existencia para dar la posibilidad de ser extensible en todos los aspectos que, didácticamente, se han considerado más relevantes.

5.2.1.- Unidad Generadora de Dictados

Otro de los trabajos futuros que se plantean para esta aplicación tiene que ver con la posibilidad, que se comentó al principio, de cargar módulos externos a la propia interfaz que ampliaran las funcionalidades de la misma.

La unidad de Generación de Dictados genera un dictado musical de acuerdo a diferentes parámetros de configuración elegida por el usuario. De esta forma se cumplirá el objetivo de crear dictados musicales adaptados al nivel correspondiente que ayude a los estudiantes a mejorar en este ámbito sin otra persona o un piano (o teclado).

La unidad generadora de dictados será capaz de mostrar una ventana externa en la que se pueda escuchar el dictado que se ha generado o visualizar el resultado.

Las características que ofrece el sistema son:

- Tipo de dictado: rítmico o melódico.
- Nivel de dificultad: 1-5 y 1-10 para la rítmica de dictado melódico.

Una vez elegidos los valores oportunos, el sistema comienza a construir el dictado.

Mientras el alumno escucha el dictado podrá escribir el resultado en el panel de edición. Posteriormente, cuando deseé visualizar el resultado, la interfaz corregirá sus errores poniendo las notas que no haya acertado en color rojo.

El funcionamiento es muy sencillo ya que la unidad generadora de dictados devuelve dos tipos de datos: un archivo MIDI (que puede ser reproducido automáticamente por la interfaz) y un vector de datos con la respuesta correcta.

De esa forma, el Sistema de Edición de Composiciones Musicales, solo deberá comparar ambos vectores, y marcar las notas que no coincidan. El método que dibuja las notas será totalmente reutilizable en este caso.

5.2.2.- Unidad Armonizadora de Melodías

Otra de las unidades que se pueden plantear para el Sistema de Edición de Composiciones Musicales es un módulo armonizador de melodías. El funcionamiento de esta unidad es aún más sencillo. El usuario introducirá gracias al editor musical, la partitura que desea armonizar.

La unidad encargada de armonizar la melodía recibirá el vector con los datos y generará n-vectores con la información de cada voz de la armonía. Asimismo, generará un archivo MIDI.

El Sistema de Edición de composiciones musicales recibirá todos los vectores, representará cada uno en una de las diferentes voces, y dará al usuario la posibilidad de reproducir el archivo MIDI, de guardar los datos en un archivo propio (para modificarlos posteriormente), e incluso la posibilidad de generar un nuevo archivo MusicXML.

5.2.3.- Entrada Manual de Notas

Aplicando las nuevas tecnologías en pantallas táctiles, sería muy interesante aplicar esta mejora en la entrada manual de notas. De esa forma, se volverían a almacenar las notas y sus características en vectores y de ahí se podrían generar los distintos tipos de archivos.

Como ya se comentó con anterioridad, el sistema de estructura de datos está adaptado a crear cualquier tipo de unidad externa siempre y cuando la forma de transmitir los datos entre unidades sea mediante vectores con las notas y sus características principales (duración, alteración tonal, etc.).

Anexos: Manual de Usuario

"La música es sinónimo de libertad, de tocar lo que quieras y como quieras, siempre que sea bueno y tenga pasión, que la música sea el alimento del amor."

Kurt D. Cobain

Para comprender bien en qué consiste la aplicación se adjunta, a modo de anexo, un manual de usuario con el que se recorrerán las distintas opciones que ofrece este Sistema de Edición de Composiciones Musicales.

1.- Menús Básicos

La estructura de la interfaz gráfica se compone de: una barra de menú con las distintas opciones en la parte superior, un panel de edición en la parte central y dos paneles con las herramientas necesarias para el dibujo de notas musicales. El resultado es el mostrado en la figura siguiente.

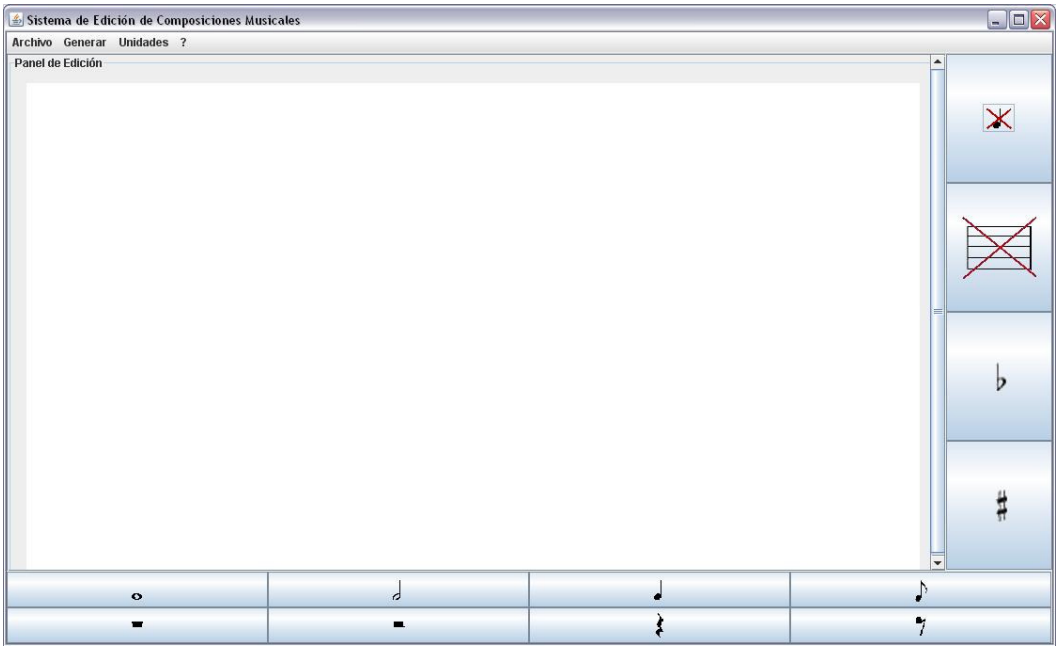

Figura 77: Interfaz Gráfica

2.- Crear Pentagramas Nuevos

Si se desea crear un pentagrama nuevo y añadirle notas se debe ir a archivo, nuevo y seleccionar el compás que se desea.

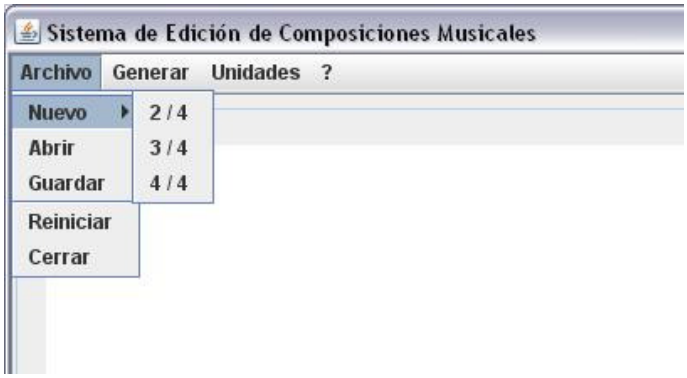

Figura 78: Archivo

Esto tendrá como resultado que en el panel de edición aparezca un pentagrama con el compás seleccionado, para el ejemplo, un compás de cuatro por cuatro.

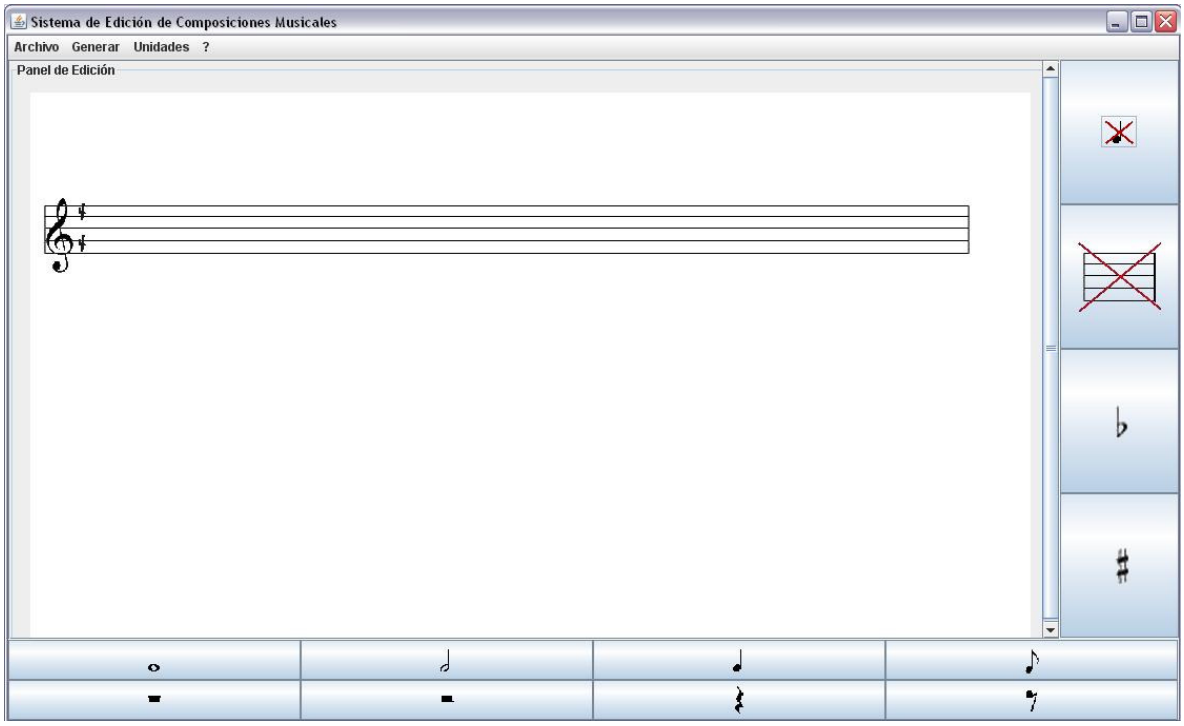

Figura 79: Interfaz con el Pentagrama

El siguiente paso será dibujar una nota en el pentagrama. Para ello primero se hará clic en alguno de los botones de la parte inferior con las distintas duraciones y a continuación hacer clic en la posición del pentagrama en la que se quiere pintar la nota.

Para el ejemplo de querer pintar una figura blanca en la nota Fa se seguirá el siguiente procedimiento.

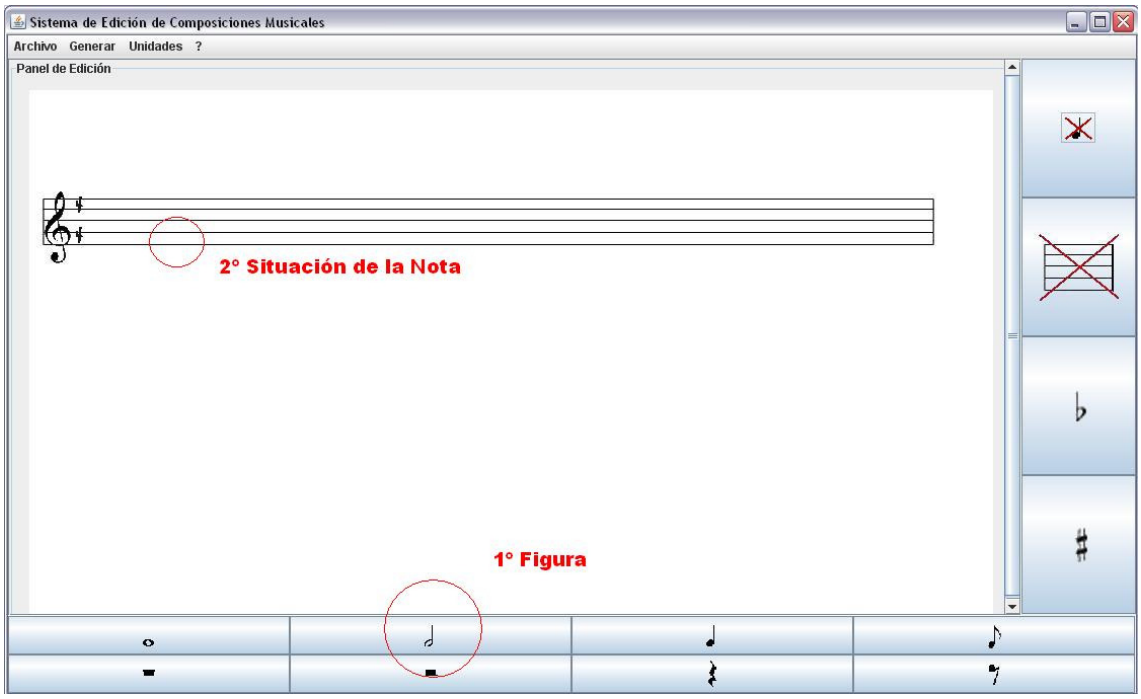

Figura 80: Pintando una nota

Y se obtendrá el resultado mostrado en la figura.

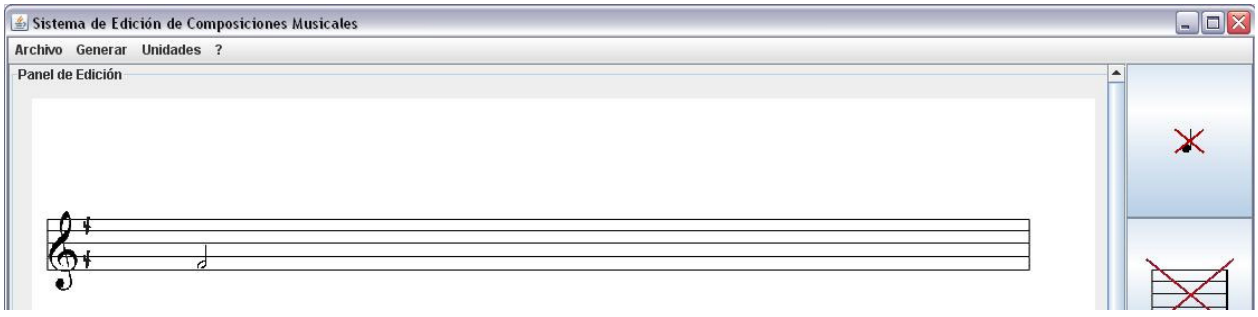

Figura 81: Resultado de pintar una nota

En el caso de querer realizar una edición más completa se contará con el panel de botones de la parte derecha de la interfaz donde se podrá borrar la última nota o el último pentagrama completo, previa confirmación.

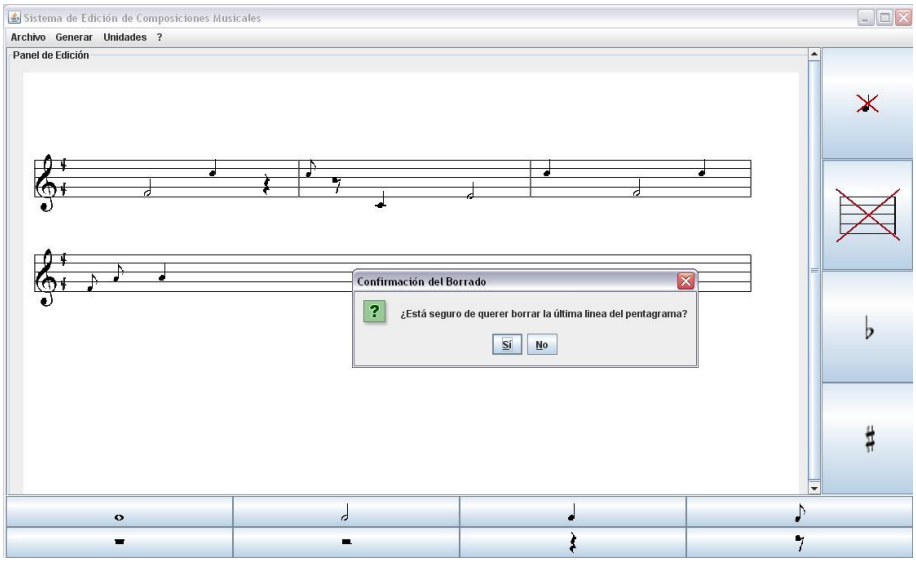

Figura 82: Confirmación del Borrado de la última línea del pentagrama

En caso de querer pintar una nota con un sostenido o bemol el procedimiento será análogo al de pintar una nota nueva sólo que antes de emplazarla se deberá pulsar la tecla del sostenido o del bemol (bien sea antes o después de pulsar la duración).

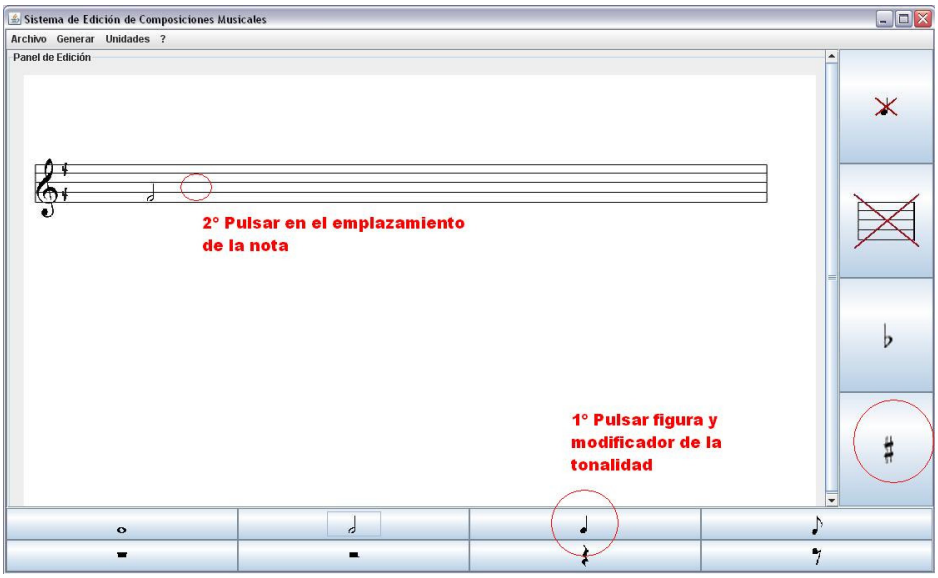

Figura 83: Proceso para pintar notas con modificadores de la tonalidad

El resultado que se obtendría para el caso anterior (pintar una figura negra con un modificador de medio tono superior, es decir un sostenido, en la nota la) sería el mostrado en la figura.

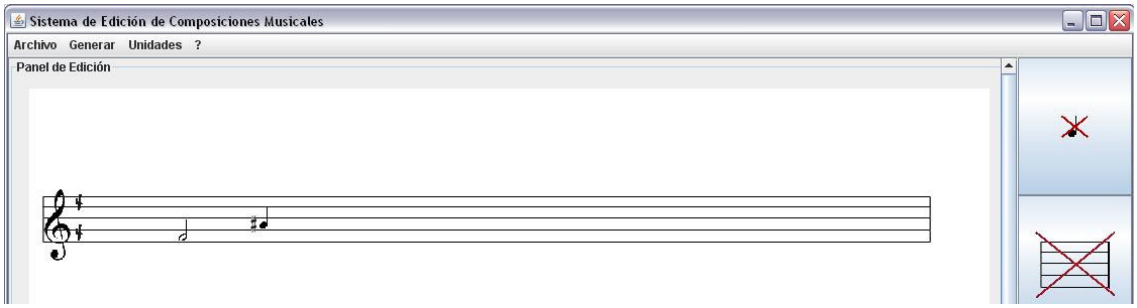

Figura 84: Resultado de pintar una nota con un sostenido

Así mismo se puede rellenar cuantas líneas de pentagrama se deseen ya que la aplicación, cuando detecte que se ha acabado una línea, pondrá automáticamente una nueva.

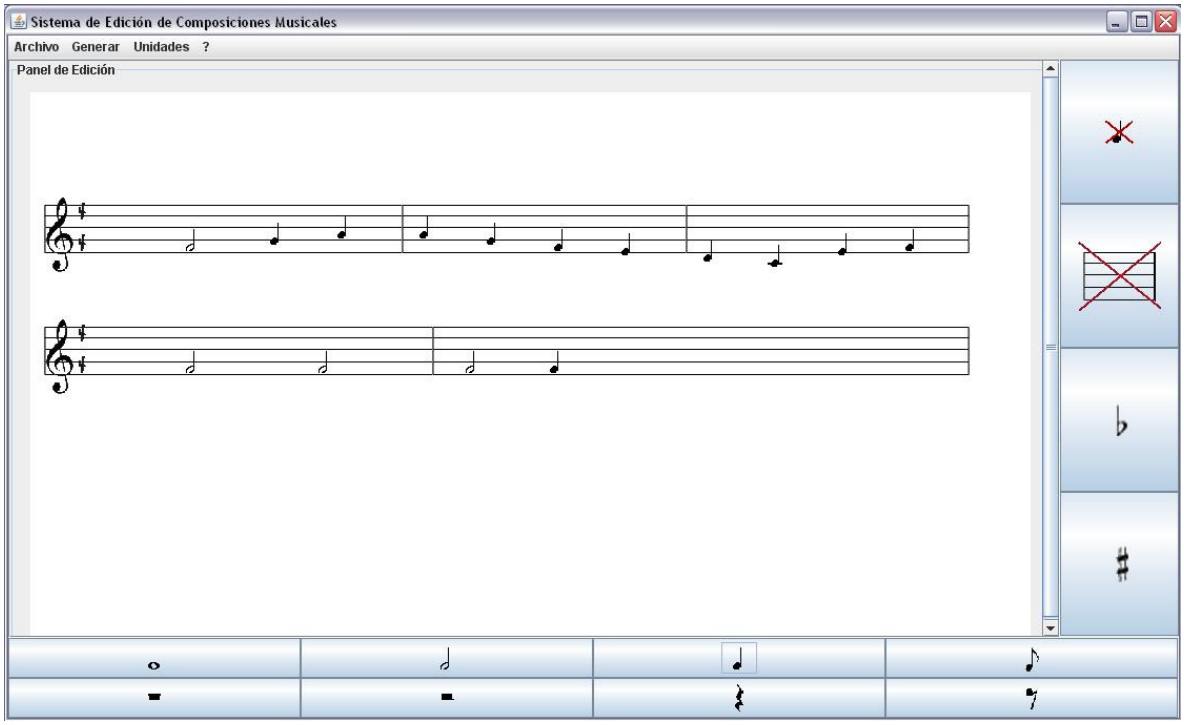

Figura 85: Breve melodía ya escrita

3.- Submenú Archivo

Se va a describir qué funcionalidades tiene dicho submenú.

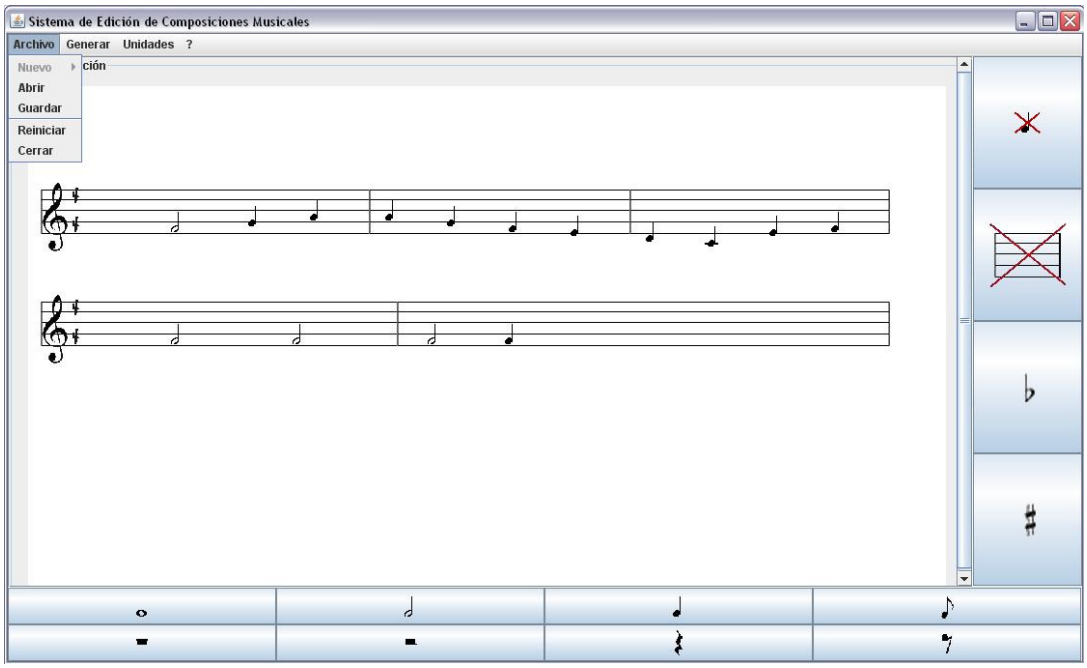

Figura 86: Submenú Archivo

Tras haber escrito una partitura se deshabilita la opción de incluir nuevos pentagramas puesto que eso lo hace automáticamente el programa. La siguiente opción que merece destacar es Guardar.

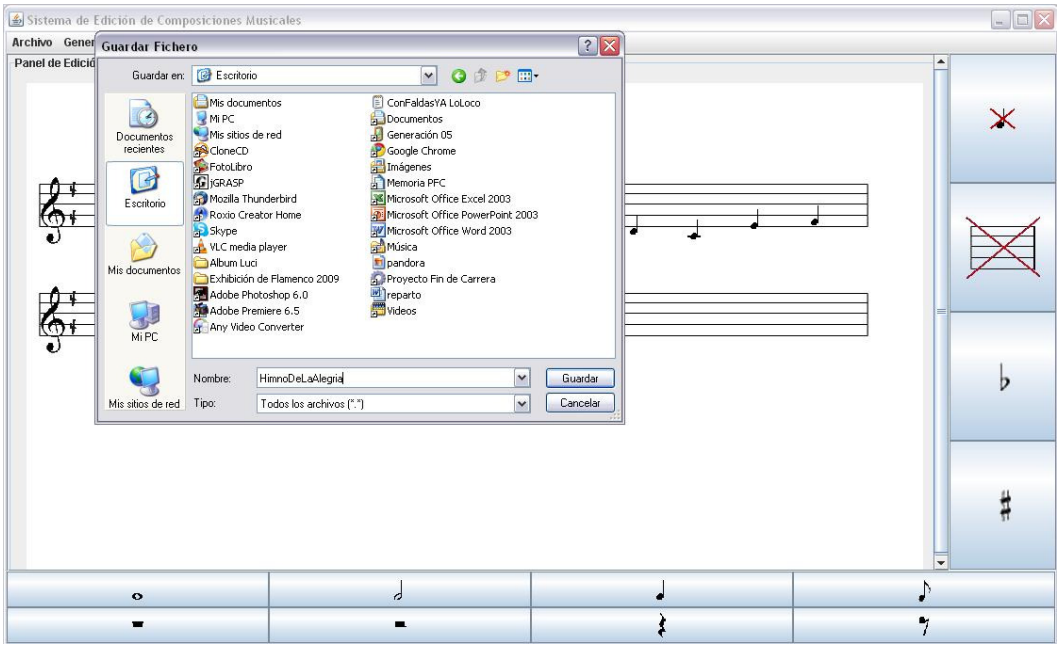

Figura 87: Guardar fichero

El archivo se guardará en el emplazamiento que elija el usuario y en formato *.mp*, un formato de archivos propio que permitirá la modificación posterior haciendo clic en archivo> abrir.

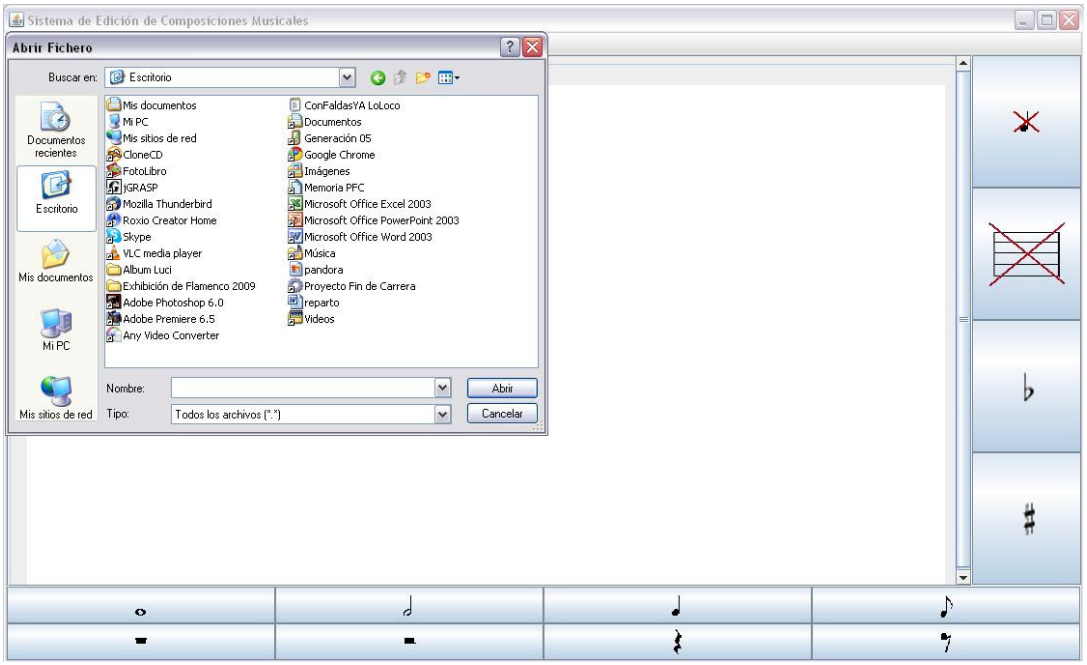

Figura 88: Abrir fichero

Después se tendrá la opción de reiniciar el panel de edición previa autorización del usuario.

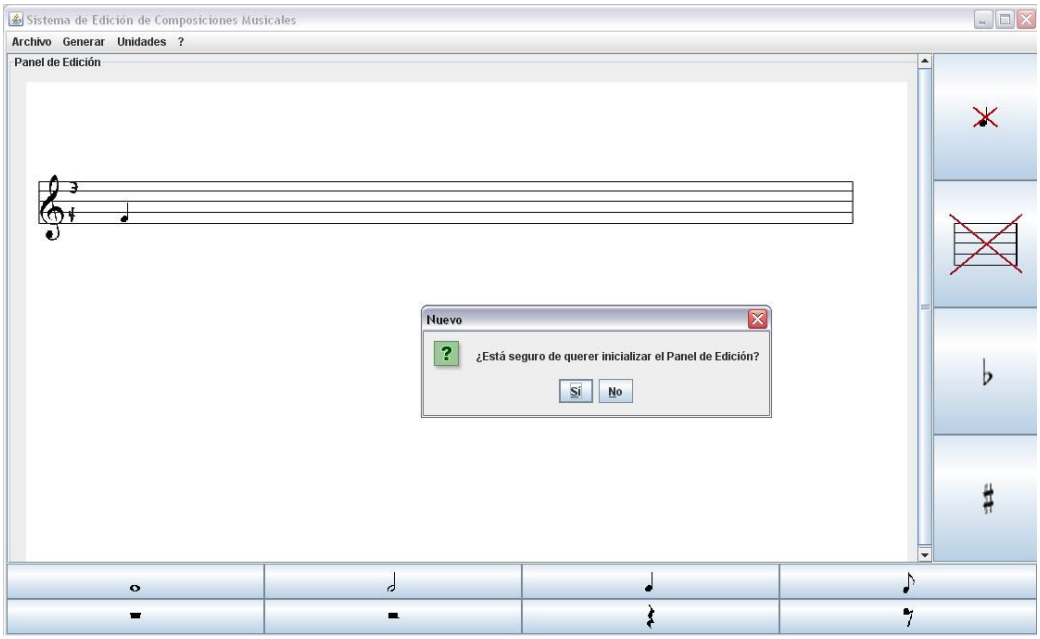

Figura 89: Reiniciar Panel de Edición

Por último existe la opción de cerrar la aplicación, como en casos anteriores.

4.- Submenú Generar

El submenú generar se ocupa básicamente de realizar dos funciones: generar archivos MusicXML y generar archivos MIDI. Dada su cercanía también se encontrará dentro del submenú la posibilidad de reproducir dichos archivos MIDI.

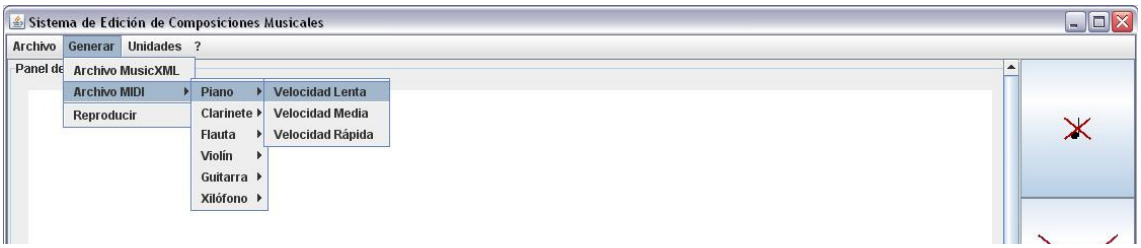

Figura 90: Submenú Generar

Inicialmente se pueden generar archivos MusicXML haciendo clic en la opción y posteriormente eligiendo dónde guardarlo.

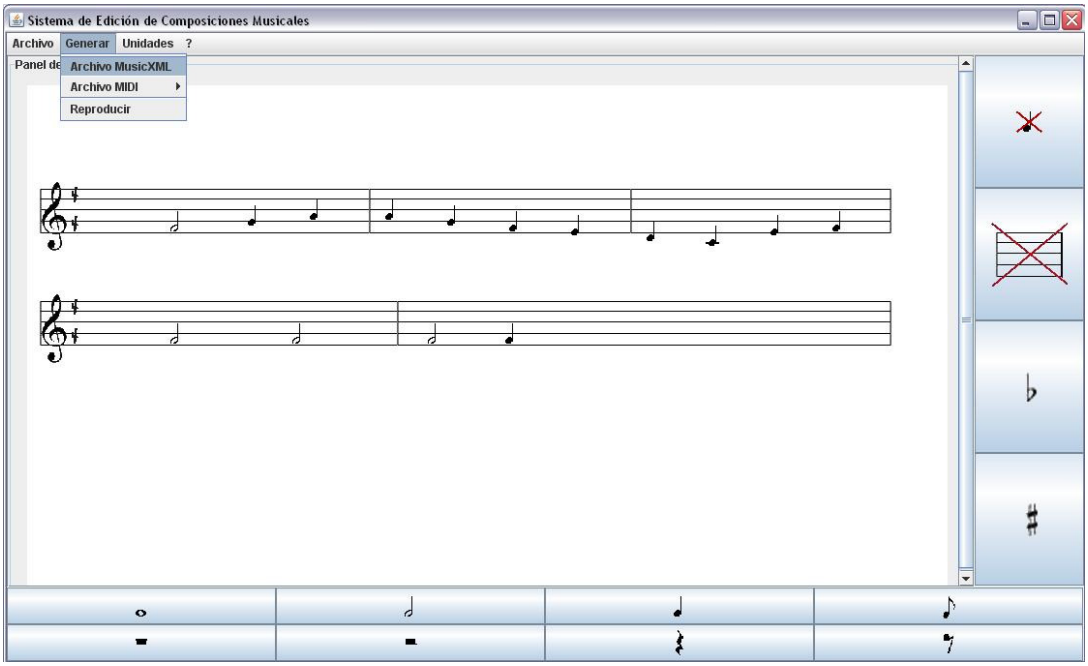

Figura 91: Opción que genera archivos MusicXML

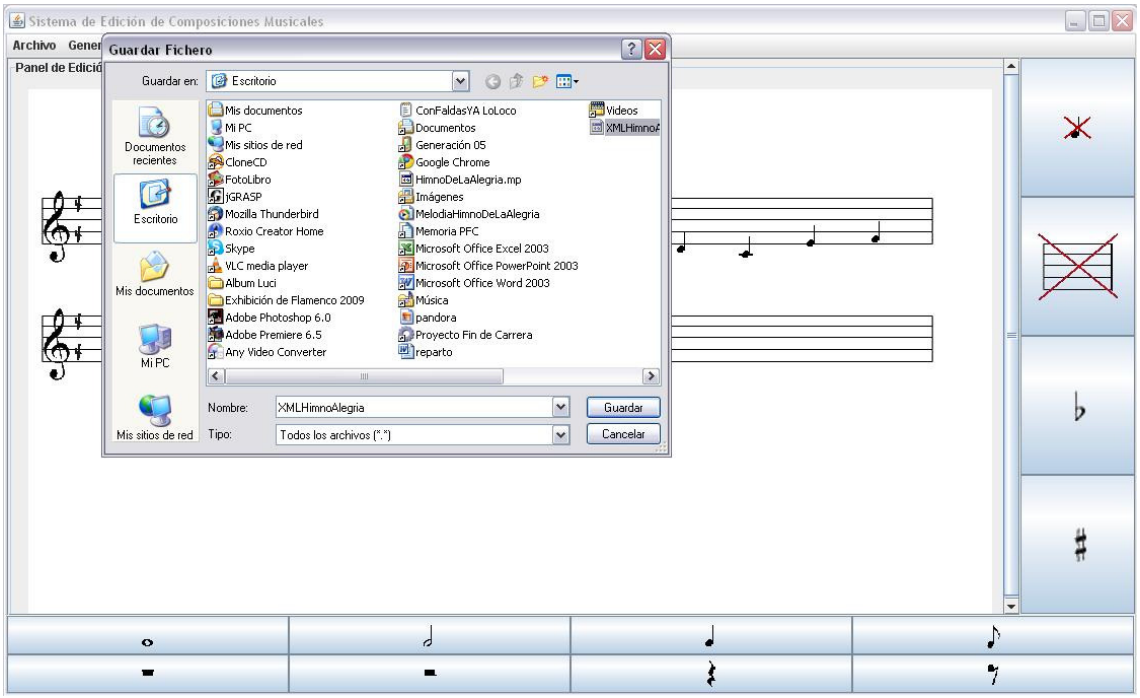

Figura 92: Selección de dónde guardar el archivo MusicXML

A continuación se pueden generar archivos MIDI para su posterior reproducción. Se pueden elegir entre instrumentos de viento: flauta y clarinete; instrumentos de cuerda: violín y guitarra; instrumentos de percusión: xilófono, y cuerda percutida: piano. Para cada instrumento se pueden seleccionar entre tres velocidades o tempos distintos: lenta, media o rápida.

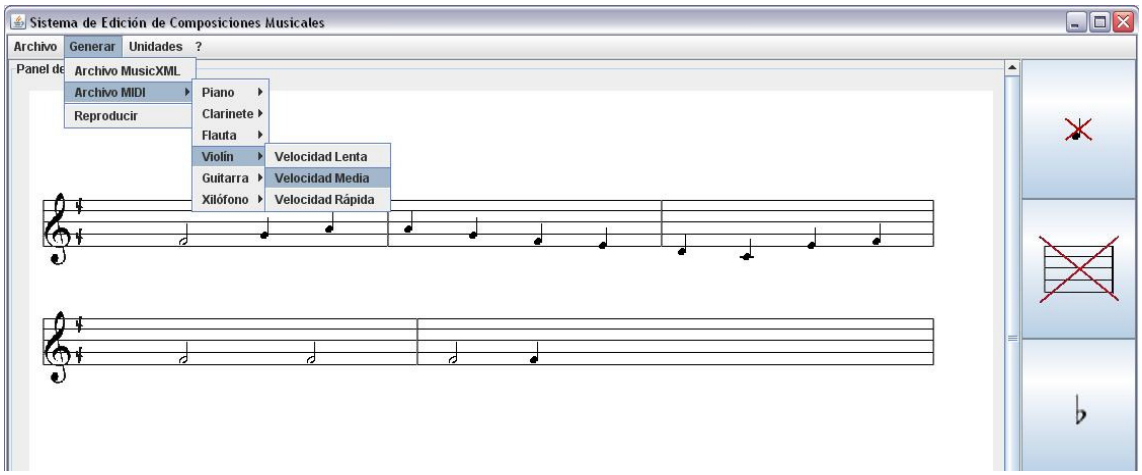

Figura 93: Visualización de distintos instrumentos y tempos

Primero se selecciona dónde se quiere guardar, según la figura:

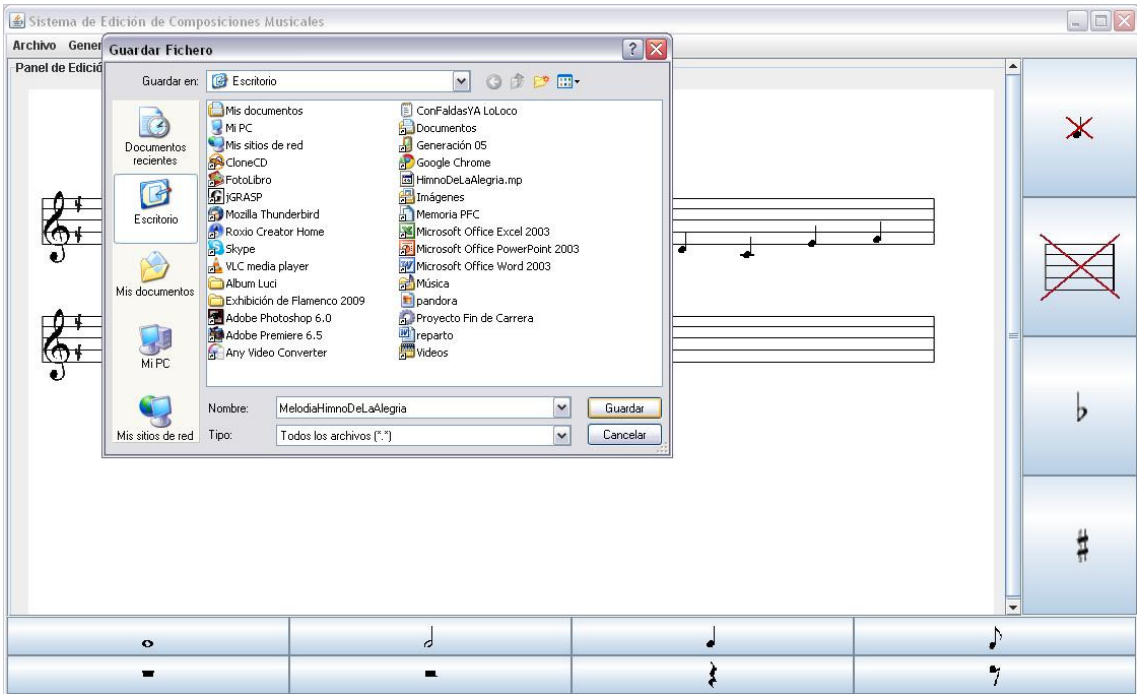

Figura 94: Selección de dónde guardar el archivo MIDI

Y por último, si es de interés para el usuario, se reproduce el archivo MIDI:

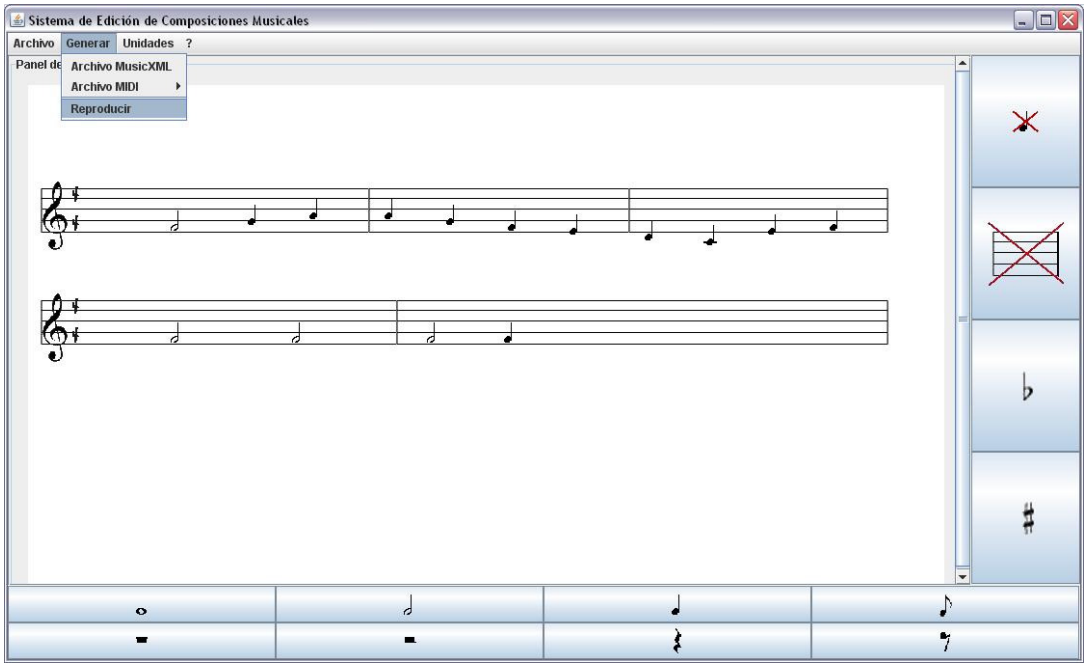

Figura 95: Opción Reproducir

5.- Submenú Unidades

El submenú unidades hace que el programa sea más extensible con la posibilidad de incluir módulos externos: el de generador de dictados musicales y el armonizador de melodías (actualmente ambos en desarrollo).

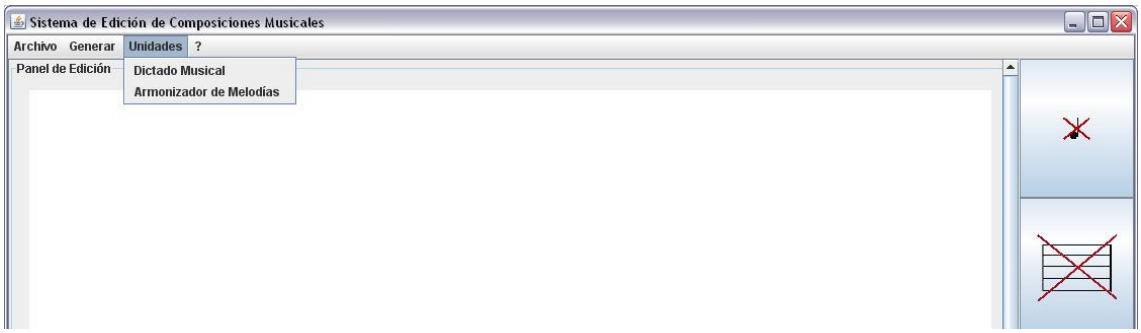

Figura 96: Submenú unidades

6.- Submenú Ayuda

El submenú Ayuda identifica al creador del programa y remite, en el apartado de ayuda, al presente manual de usuario.

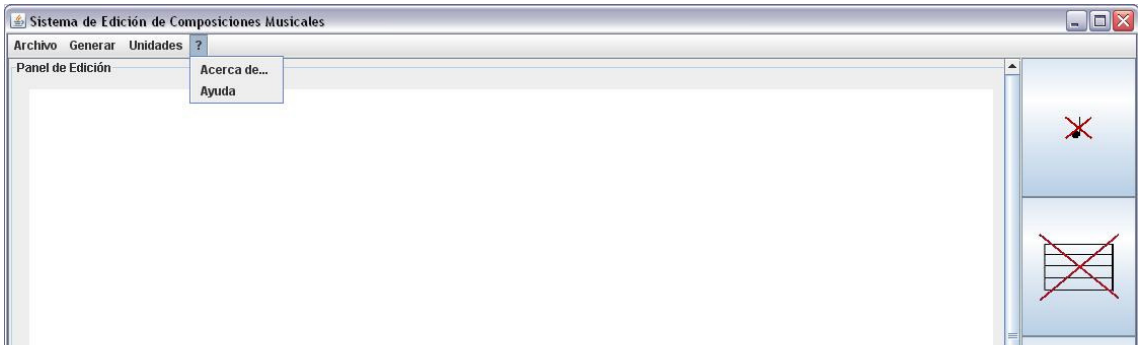

Figura 97: Submenú de ayuda

Bibliografía

"Los músicos no se retiran: paran cuando no hay más música en su interior."

Louis Armstrong

Para el desarrollo del Proyecto Fin de Carrera "Sistemas de Edición de Composiciones Musicales se ha utilizado la siguiente documentación que se cita por orden alfabético:

[1] American Music Conference, www.amc-music.com (visitada el 25/5/2009).

[2] API Specifications of Java 2 Platform, SE v 1.4.2, http://java.sun.com/j2se/1.4.2/docs/api/ (visitada 1/8/2009).

[3] API Specifications of JMusic, http://jmusic.ci.qut.edu.au/jmDocumentation/index.html (visitada el 30/5/09).

[4] Arteaga Aldana, Eduardo "La informática aplicada a la edición musical", Artículo de la Universidad Carlos III de Madrid.

[5] BBC, http://news.bbc.co.uk/2/hi/technology/7458479.stm (visitada el 28/5/2009).

[6] Blanco, Xavier, "El protocolo MIDI", http://www.hispasonic.com/revista/protocolo-midi (visitada el 22/6/2009).

[7] Bou Bauzá , Gillem, "El guión multimedia". Anaya Multimedia, 1997.

[8] Diccionario de la Real Academia de la Lengua Española, Vigésima Primera Edición, Espasa Calpe, 1994.

[9] Denemo Open Source Music Notation, http://www.denemo.org/ (visitada el 10/8/2009).

[10] Finale Music Compositicon, http://www.finalemusic.com/ (visitada el 10/8/2009).

[11] Galvez Rojas y Mora Mata, "Tutorial sobre compiladores", http://www.lcc.uma.es/~galvez/Compiladores.html (visitada el 20/6/2009).

[12] G. F. Software, http://www.gfsoftware.com/ (visitada el 2/8/2009).

[13] Gran Enciclopedia Larousse, (tomos: 12, 16 y 24), Editorial Planeta, 1990.

[14] GVOX, http://www.gvox.com (visitada el 5/8/2009).

[15] Ibáñez y Cursá, "Cuadernos de Lenguaje", Real Musical 1994

[16] JMusic Official Web http://jmusicsic.ci.qut.edu.au (visitada el 5/6/2009).

[17] J. Sánchez, G. Huecas, B. Fernández, P. Moreno, "Java 2, Iniciación y Referencia", Editorial Mc Graw Hill, 2005.

[18] Lévense, Eric, "Historia de los Lenguajes de Programación", http://www.levenez.com/lang/ (visitada el 15/6/2009).

[19] MagicScore Music Notation Software, http://www.musicaleditor.com/ (visitada el 7/8/2009).

[20] Melbourne School of Engineering, Department of Computer Science and Software Engineering, CSIRAC http://www.csse.unimelb.edu.au/dept/about/csirac/ (visitada el 28/5/2009).

[21] MIDI Manufacturers Association, http://www.midi.org/ (visitada el 23/6/2009).

[22] MuseScore, http://www.musescore.org/es (visitada el 6/8/2009).

[23] MusicXML v. 2.0 Official Web http://www.recordare.com/xml.html (visitada el 30/6/2009).

[24] NoteEdit, http://noteedit.berlios.de/ (visitada el 12/8/2009).

[25] Power Tab, http://www.power-tab.net/ (visitada el 1/8/2009)

[26] Ray, Kristof, "Diseño interactivo". Anaya Multimedia, 1998.

[27] Sibelius, http://www.sibelius.com (visitada el 6/8/2009).

[28] Tejada, J. "Música e Inteligencia Artificial". Departamento de expresión artística. Universidad de la Rioja http://www.cmrioja.es/musica/musia.html (visitada el 8/8/2009).

[29] Turina, Joaquín, "Enciclopedia Abreviada de la Música", Biblioteca Nueva, 1996.

[30] Universidad de Princetown, Departament of Computer Science sobre "*International Conference on New Interfaces for Musical Expression*", http://soundlab.cs.princeton.edu/publications/ (visitada el 20/5/2009).

[31] Vivaldi Studio, http://www.vivaldistudio.com/ (visitada el 5/8/2009).

[32] Web Exception, http://www.alegsa.com.ar/Notas/2.php (visitada el 20/5/2009).